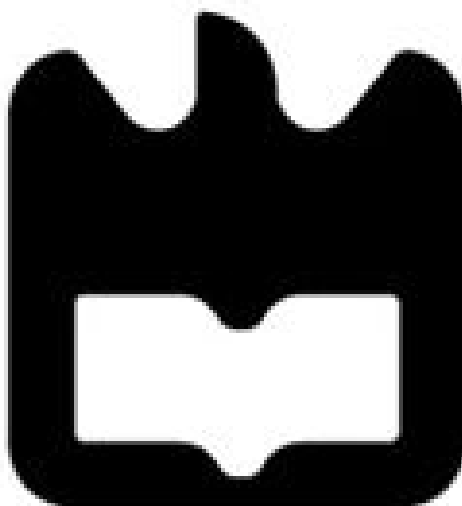




**João Paulo de
Jesus Aparício**

Integração da *Cloud* com a rede do Operador





**João Paulo de
Jesus Aparício**

Integração da *Cloud* com a rede do Operador

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica da Professora Doutora Susana Sargento, Professora Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática de Aveiro, e do Mestre Jorge Carapinha, Engenheiro do Departamento de Coordenação Tecnológica e Inovação Exploratória da Portugal Telecom Inovação.

o júri / the jury

presidente

Professor Doutor Rui Luís Andrade Aguiar

Professor Associado C/ Agregação da Universidade de Aveiro

Vogais / examiners committee

Professor Doutor Paulo Alexandre Ferreira Simões

Professor Auxiliar do Departamento de Engenharia Informática da
Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Professora Doutora Susana Isabel Barreto de Miranda Sargento

Professora Auxiliar da Universidade de Aveiro

Mestre Jorge Manuel dos Santos Correia Carapinha

Investigador Sénior da Portugal Telecom Inovação

Agradecimentos / acknowledgements

Esta dissertação representa o fim do meu ciclo universitário, cinco anos árduos que muito contribuíram para o meu desenvolvimento, tanto intelectual como humano. Ao longo da dissertação, foram várias as pessoas que me ajudaram e contribuíram para que tivesse o maior sucesso possível.

Quero agradecer à Professora Doutora Susana Sargento e ao Engenheiro Jorge Carapinha da PT Inovação, que me orientaram ao longo deste trabalho, pela responsabilidade, inteligência, profissionalismo e atitude com que o fizeram.

Ao Mestre João Soares quero agradecer todo o apoio que me prestou, desde discussões científicas a conselhos que visaram sempre em melhorar o meu trabalho; também quero agradecer aos meus companheiros do laboratório de redes 2 do Instituto de Telecomunicações pelo apoio e paciência dia-a-dia.

À minha Mulher por toda a sua paciência, encorajamento e apoio durante este período e à minha futura Filha, que ainda sem ter nascido, já ouviu e deu força ao seu pai para completar esta etapa.

Por fim, quero agradecer aos meus Pais e Irmão por todo o apoio e confiança demonstrada nas minhas capacidades ao longo desta importante e especial etapa da minha vida.

palavras-chave

Cloud, Mapeamento, Redes Virtuais, Virtualização, SDN

resumo

A utilização das aplicações e as suas formas de comunicar mudaram muito com a proliferação do acesso à *Internet*. Com esta alteração muitas das aplicações passaram a estar acomodadas em equipamento do fornecedor em vez do equipamento do utilizador. *Cloud computing* (CC) é o conceito que veio "patrocinar" ainda mais esta mudança. Hoje o fornecimento destes serviços é suportado pelo serviço *Best Effort* que a *Internet* disponibiliza. Este é um modelo viável para alguns serviços, mas simplesmente inaceitável para outros (por exemplo, transmissões de vídeo). No sentido de colmatar esta lacuna, existe uma grande aposta nos serviços integrados de *cloud* e de rede. A este paradigma denominamos de *Cloud Networking*. Este paradigma requer o estabelecimento a pedido e um controlo e gestão automática de recursos de rede e *cloud*, em que a virtualização de rede e de recursos *cloud* é uma peça fundamental, não só pela sua facilidade de migração de recursos virtuais entre diferentes máquinas físicas, mas também pela flexibilidade do estabelecimento de aplicações e serviços diferentes. Neste contexto o recente conceito de *software-defined networks* (SDN) pode vir ajudar a melhorar o desempenho dos serviços disponibilizados na *cloud*.

Assim, esta dissertação tem dois objetivos. O primeiro visa trabalhar em mecanismos de gestão de recursos de *cloud* e rede de uma forma integrada. Concretamente esta dissertação propõe um algoritmo de mapeamento, bem como um mecanismo de reconfiguração de *links* por forma a otimizar a alocação de recursos e aumentar a aceitação de pedidos. O segundo objetivo passou por criar um bloco funcional de decisão de mapeamento e reconfiguração que se encaixa numa arquitetura SDN. Este bloco é responsável por receber, analisar e mapear pedidos de serviços de conectividade sobre uma rede *Openflow*. Os algoritmos usados neste componente têm em conta as considerações alcançadas na primeira parte da dissertação.

Os resultados obtidos permitem verificar que o algoritmo de mapeamento de recursos de *cloud* e rede, bem como o mecanismo de reconfiguração de *links*, proporcionam um desempenho significativamente superior aos algoritmos do estado da arte, com uma maior aceitação e ganhos à custa de uma utilização inferior dos recursos de rede, e com um consumo energético inferior. O bloco funcional fecha o ciclo básico de controlo da arquitetura SDN para a receção e tratamento de serviços de conectividade. O estudo global dá uma noção do desempenho geral da arquitetura completa, e o estudo individual das diferentes partes do bloco funcional permite perceber quais as partes dentro do componente proposto que deverão ser melhoradas no futuro.

keywords

Cloud, Mapping, Virtual Network, Virtualization, SDN

abstract

The use of applications and their ways of communicating have greatly changed with the proliferation of Internet access. With this, these applications have come to be accommodated in the equipment supplier rather than the user equipment. Cloud computing (CC) is the concept that came and “sponsored” even more this change. Today the supply of these services is supported by the Best Effort service that the Internet provides. This is a feasible model for some services, but it is simply unacceptable to others (i.e video streams). In order to fill this gap, there is a big bet in integrating cloud and network elements together. To this paradigm we call Cloud Networking. This paradigm requires the establishment, application monitoring and automatic management of network and cloud resources, where both network and cloud virtualization are a key role, not only because of its easiness migration of virtual resources between physical machines, but also by the flexibility of setting different applications and services. In this context, the software-defined networks (SDN) can help improve the performance of the available cloud services.

This Dissertation has two objectives. The first one is to work on mechanisms of resource management and cloud network in an integrated way. Specifically, this Dissertation proposes a mapping algorithm as well as a mechanism for reconfiguring links to optimize resource allocation and increase the acceptance of applications. The second goal is the creation of a functional component for mapping decision and reoptimization that fits in an SDN framework. This component is responsible for receiving, analyzing and mapping requests for connectivity services over an OpenFlow network. The algorithms used in this component take into account the considerations achieved with the first part of the Dissertation.

The results lead us to conclude that the proposed mapping algorithm for cloud and network resources, as well as the mechanism for reconfiguring links, achieve a performance significantly superior to the state of art algorithms, with a higher acceptance and gains at the expense of a lower utilization of network resources, and a lower energy consumption. The functional component closes the control basic cycle of the SDN framework to the reception and treatment of connectivity services. The global performance study gives perception of the general performance of the complete SDN solution, and the individual study of the different parts of the functional component allows us to understand the parts inside the proposed component that should be improved in the future.

Contents

CONTENTS	I
LIST OF FIGURES.....	V
LIST OF TABLES	VII
ACRONYMS.....	IX
1. INTRODUCTION	1
1.1. MOTIVATION	1
1.2. PURPOSE	2
1.3. CONTRIBUTION	2
1.4. DISSERTATION OUTLINE.....	3
2. STATE OF THE ART.....	5
2.1. INTRODUCTION	5
2.2. CLOUD COMPUTING.....	5
2.2.1. CONCEPT OVERVIEW	5
2.2.2. SERVICES.....	6
2.2.3. MODELS	8
2.2.4. COMMERCIAL OFFERS AND PROVIDERS.....	9
2.2.5. CLOUD MANAGEMENT PLATFORMS	10
2.2.6. THE NETWORK'S ROLE	12
2.2.7. "CLOUD NETWORKING"	13
2.3. NETWORKING	14
2.3.1. INTRODUCTION	14
2.3.2. NETWORK VIRTUALIZATION	15
2.3.2.1. CONCEPT OVERVIEW	15
2.3.2.2. ARCHITECTURE.....	15
2.3.2.3. PLATFORMS & COMMERCIAL OFFERS	16
2.3.3. SOFTWARE DEFINED NETWORKING	17
2.3.3.1. CONCEPT OVERVIEW	17
2.3.3.2. ARCHITECTURE.....	17
2.3.3.3. OPENFLOW	18
2.3.3.4. CONTROLLER PLATFORMS	18

2.3.3.5. COMMERCIAL OFFERS	19
2.4. RESOURCE MANAGEMENT ALGORITHMS	20
2.4.1. CLOUD	20
2.4.2. NETWORK	21
2.4.3. CLOUD AND NETWORK	22
2.5. SUMMARY	23
3. CLOUD NETWORK MAPPING ALGORITHM	25
3.1. INTRODUCTION	25
3.1.1. <i>Discrete event simulator</i>	26
3.2. ALGORITHM DESCRIPTION	26
3.3. CANDIDATES LIST	29
3.3.1. <i>Candidates first selection try</i>	30
3.3.2. <i>Check Routine</i>	30
3.3.3. <i>Performance Evaluation</i>	32
3.4. DECISION PROCESS	34
3.4.1. <i>Node Stress</i>	35
3.4.2. <i>HOP_Dijkstra</i>	35
3.4.3. <i>Performance Evaluation</i>	38
3.5. OVERALL PERFORMANCE EVALUATION	40
3.6. CONCLUSIONS	47
4. LINK REOPTIMIZATION STRATEGY	49
4.1. INTRODUCTION	49
4.2. ALGORITHM DESCRIPTION	50
4.3. STRATEGIES	51
4.3.1. BANDWIDTH GROUPING	51
4.3.1.1. PERFORMANCE EVALUATION	52
4.3.2. DIFFERENT APPROACHES	55
4.3.2.1. PERFORMANCE EVALUATION	56
4.4. OVERALL PERFORMANCE EVALUATION	59
4.4.1. PERIODICITY	59
4.4.2. HEURISTIC COMPARISON	61
4.5. CONCLUSIONS	65
5. SOFTWARE-DEFINED-NETWORK ORIENTED SERVICE MANAGER	67
5.1. INTRODUCTION	67
5.2. SDN FRAMEWORK ARCHITECTURE	67
5.3. SERVICE MANAGER IMPLEMENTATION	69
5.3.1. SERVICE HANDLER	70
5.3.2. DATABASE	72
5.3.3. FLOW HANDLER	73
5.4. PERFORMANCE EVALUATION	75
5.5. CONCLUSION	82
6. CONCLUSIONS	83

6.1.1. FINAL CONCLUSIONS	83
6.1.2. FUTURE WORK	84
BIBLIOGRAPHY	87

List of Figures

FIGURE 2.1 – CLOUD COMPUTING SERVICES [3].....	7
FIGURE 2.2 – <i>OPENSTACK</i> ARCHITECTURE [15].....	11
FIGURE 2.3 – “NETWORKING SETUP IN A ZONE” [21]	12
FIGURE 2.4 – THE EVOLUTIONARY PROCESS OF CLOUD AND NETWORK [28].....	14
FIGURE 2.5 – CREATION OF VIRTUAL NETWORKS [30]	15
FIGURE 2.6 – SDN ARCHITECTURE [36]	18
FIGURE 3.1 – VI ACCEPTANCE AND REVENUE RATIO - CANDIDATES LIST	33
FIGURE 3.2 – PHYSICAL INFRASTRUCTURE LINK OCCUPATION RATIO - CANDIDATES LIST	33
FIGURE 3.3 – PHYSICAL INFRASTRUCTURE USAGE - CANDIDATES LIST	34
FIGURE 3.4 – VIRTUAL LINK PATH FINDING	36
FIGURE 3.5 – VIRTUAL LINK PATH FINDING FOR THE BOTH CANDIDATES A AND G	37
FIGURE 3.6 – VI ACCEPTANCE AND REVENUE RATIO - DECISION PROCESS	38
FIGURE 3.7 – PHYSICAL INFRASTRUCTURE LINK OCCUPATION RATIO - DECISION PROCESS	39
FIGURE 3.8 – PHYSICAL INFRASTRUCTURE USAGE - DECISION PROCESS	40
FIGURE 3.9 – VI ACCEPTANCE AND REVENUE RATIO	41
FIGURE 3.10 – VI ACCEPTANCE AND REVENUE RATIO - REQUEST RATE VARIATION	41
FIGURE 3.11 – PHYSICAL INFRASTRUCTURE LINK OCCUPATION RATIO	42
FIGURE 3.12 – PHYSICAL INFRASTRUCTURE USAGE	43
FIGURE 3.13 – PHYSICAL INFRASTRUCTURE USAGE - REQUEST RATE VARIATION	43
FIGURE 3.14 – PHYSICAL INFRASTRUCTURE OCCUPATION (SUBSTRATE 15 NODES).....	44
FIGURE 3.15 – PHYSICAL INFRASTRUCTURE OCCUPATION (SUBSTRATE 25 NODES).....	45
FIGURE 3.16 – PHYSICAL INFRASTRUCTURE OCCUPATION (SUBSTRATE 35 NODES).....	45
FIGURE 3.17 – PHYSICAL INFRASTRUCTURE OCCUPATION (SUBSTRATE 45 NODES).....	46
FIGURE 3.18 – PHYSICAL INFRASTRUCTURE OCCUPATION (SUBSTRATE 55 NODES).....	46
FIGURE 4.1 – VI ACCEPTANCE AND REVENUE RATIO & PHYSICAL INFRASTRUCTURE LINK OCCUPATION RATIO - PERIODICITY	60
FIGURE 4.2 – AVERAGE NUMBER OF CHANGES ON VIs & PHYSICAL INFRASTRUCTURE USAGE - PERIODICITY	60
FIGURE 4.3 – VI ACCEPTANCE AND REVENUE RATIO	62
FIGURE 4.4 – VI ACCEPTANCE AND REVENUE RATIO - REQUEST RATE VARIATION	62
FIGURE 4.5 – PHYSICAL INFRASTRUCTURE LINK OCCUPATION	63
FIGURE 4.6 – PHYSICAL INFRASTRUCTURE USAGE	64
FIGURE 4.7 – PHYSICAL INFRASTRUCTURE USAGE - REQUEST RATE VARIATION	64
FIGURE 5.1 – SDN FRAMEWORK ARCHITECTURE	68

FIGURE 5.2 – SERVICE MANAGER ARCHITECTURE	70
FIGURE 5.3 – SERVICE MANAGER DATABASE.....	72
FIGURE 5.4 – SERVICE RECEIVER PERFORMANCE FULL MESH	76
FIGURE 5.5 – SERVICE RECEIVER PERFORMANCE MULTICAST	76
FIGURE 5.6 – TIME CONSUMPTION FOR THE TRANSLATION, STORAGE AND COMPLETION OF THE FLOWS (FULL MESH).....	77
FIGURE 5.7 – TIME CONSUMPTION FOR THE TRANSLATION, STORAGE AND COMPLETION OF THE FLOWS (MULTICAST)	77
FIGURE 5.8 – SUBSTRATE NETWORK MOMENTS OF PERFORMANCE EVALUATION	78
FIGURE 5.9 – TIME TO ACTIVATE THE CONNECTIVITY SERVICES.....	80
FIGURE 5.10 – TIME TO REACT TO A LINK REMOVAL (SUBSTRATE NETWORK).....	80
FIGURE 5.11 – TIME TO REACT TO A LINK ADDITION (SUBSTRATE NETWORK)	81

List of Tables

TABLE 2.1 – COMMERCIAL SWITCHES THAT SUPPORT THE <i>OPENFLOW</i> PROTOCOL.....	19
TABLE 3.1 – PHYSICAL AND VIRTUAL NETWORK RESOURCE PARAMETERS	26
TABLE 3.2 – BASE ALGORITHM DECISIONS - EXAMPLE 1	31
TABLE 3.3 – ALGORITHM WITH CHECK ROUTINE DECISIONS - EXAMPLE 2	31
TABLE 3.4 – BASE ALGORITHM DECISIONS - EXAMPLE 3	31
TABLE 3.5 – ALGORITHM WITH CHECK ROUTINE DECISIONS - EXAMPLE 4	32
TABLE 4.1 – INDIVIDUAL PERFORMANCE - GROUPS	53
TABLE 4.2 – VI ACCEPTANCE AND REVENUE RATIO - GROUPS	54
TABLE 4.3 – PHYSICAL INFRASTRUCTURE LINK OCCUPATION RATIO - GROUPS	54
TABLE 4.4 – PHYSICAL INFRASTRUCTURE USAGE - GROUPS	55
TABLE 4.5 – INDIVIDUAL PERFORMANCE - APPROACHES.....	57
TABLE 4.6 – VI ACCEPTANCE AND REVENUE RATIO - APPROACHES.....	58
TABLE 4.7 – PHYSICAL INFRASTRUCTURE LINK OCCUPATION RATIO - APPROACHES	58
TABLE 4.8 – PHYSICAL INFRASTRUCTURE USAGE - APPROACHES.....	59
TABLE 5.1 – URL TO RECEIVE THE CONNECTIVITY SERVICES TO ACTIVATE	71
TABLE 5.2 – URL TO INTERACT WITH THE CREATED CONNECTIVITY SERVICES	71
TABLE 5.3 – URL TO RECEIVE THE SUBSTRATE NETWORK INFORMATION.....	74
TABLE 5.4 – URL TO RECEIVE THE ALARMS REGARDING THE OPERATION OF THE SUBSTRATE NETWORK.....	74
TABLE 5.5 – PHYSICAL MACHINE FEATURES	75
TABLE 5.6 – TESTBED SPECIFICATION	78

Acronyms

Amazon EC2	Amazon Elastic Compute Cloud
Amazon S3	Amazon Simple Storage Service
Amazon VPC	Amazon Virtual Private Cloud
API	Application Programming Interface
AWS	Amazon Web Services
BGP	Border Gateway Protocol
BoD	Bandwidth on Demand
CABO	Concurrent Architectures are Better than One
CAPEX	Capital expenses
CC	Cloud Computing
CI	Confidence Interval
CP	Content Providers
CPU	Central Processor unit
CRM	Customer Relationship Management
DRE	Distributed Real-time Embedded
EPCB	Extended Power Consumption-Based
GENI	Global Environment for Network Innovation
GO	Golang
GUI	Graphical user interface
HDD	Hard Drive Disk
IaaS	Infrastructure as a Service
IP	Internet Protocol
ISP	Internet Service Providers

IT Information Technology

LAN Local-Area Network

MAC Media Access Control Address

MILP Mixed Integer Linear Programming

MPLS Multiprotocol Label Switching

NaaS Network as a Service

NIST National Institute of Standards and Technology

NR Neighborhood Resource Availability

OPEX Operation expenses

OVS Open Virtual Switch

PaaS Platform as a Service

PCB Power Consumption-Based

PDA Personal Digital Assistant

PHP Hypertext Preprocessor

QoS Quality of Service

RAM Random Access Memory

RR Round-Robin

RTT Round-Trip-Times

SaaS Software as a Service

SDN Software-Defined Network

SDK Software Development Kit

SLA Service Level Agreement

SNAC Simple Network Access Control

TOPSIS Technique for Order of Preference by Similarity to Ideal Solution

TRB Transmission Rate-Based

URL Uniform Resource Locator

VI Virtual Infrastructure

VN Virtual Network

VM Virtual Machine

VPLS Virtual Private Lan Service

VPN Virtual Private Network

WAN Wide-Area Network

WSGI Web Server Gateway Interface

1. Introduction

1.1. Motivation

Evolution brings more evolution. The emergence of the Cloud Computing (CC) is one concrete example of the previous sentence. Before, when the *Internet* was not so popular, the mechanism used to communicate and run applications was completely different from the used nowadays. Back then, applications needed to be installed or hosted in the user computer. The evolution and popularity of the *Internet* allowed new services to appear. Now it is possible to run the applications far away from the computer of the user but still allowing him to take the most out of the application as if it was in his personal computer. The place where the applications are hosted is sometimes unknown, and the consumed computing resources become irrelevant for the user, since for the user what matters is that the service is always available. This hosting place is called the *cloud*. CC's potential overcame all the expectations and is changing the traditional business models and Information Technology (IT) companies operations.

The deployment of the *cloud* services is done via dedicated service providers, such as the *Amazon, Oracle, Google, Salesforce.com, Microsoft* and *Portugal Telecom*. Now we have services which can store large amounts of space (Gigabytes) such as *Google Docs* through the *Internet*. One problem that appeared with these new services was the conditions of the access. The *Internet* provides those services in a best effort way. For some kind of services this is completely suitable, but there are services that simply do not accept that. In order to solve this problem, providers started to invest in the integration of cloud services and network services. This paradigm is referred to as *Cloud Networking*.

The *Cloud Networking* paradigm requires the establishment, monitoring and automatic management of network resources and cloud resources. Here, network virtualization and cloud resources virtualization perform a key role, not only for its ease of migration of resources between different physical machines, but also by the flexibility of setting different applications and services. In this context, the recent concept of SDN can help to improve the performance of the services available in the cloud.

1.2. Purpose

The aim of this Dissertation is to work on the integration of cloud resources and network resources. One of the challenges that arise in this integration scenario is the process of mapping those resources into the operator's physical infrastructure.

In this sense we propose a mapping algorithm that maps cloud and network resources, which we denominate as Virtual Infrastructure (VI), with the objective of maximizing the embedding ratio as well as the provider's profit. In addition, we also propose a link reoptimization mechanism to optimize the resource allocation and increase the embedding of VIs.

Related with SDN, a functional component is designed to receive, handle and map connectivity services above an *Openflow* network. Furthermore, this component performs the management of those services in order to assure the good operation of the substrate network. The algorithms implemented in this component take into account the considerations of the results of the mapping algorithm proposed in the first part of the Dissertation.

In order to analyze the quality of the proposed approaches, performance tests are done to each individual part: in the case of the mapping algorithms, they are compared with previous works; for the SDN component, individual performance tests are performed followed by performance tests of our complete SDN framework.

1.3. Contribution

As a result of the accomplishment of the proposed objectives, this Dissertation contributes with a more efficient mapping algorithm that maximizes the gains of the operators, thus allowing more VI to be allocated in their physical infrastructure. More than that, it also reduces the energy consumption costs of the physical infrastructure.

The link reoptimization algorithm enhances more those features. Through the comparison of different approaches, it is noticeable the bottleneck of the VI embedding process and which are the factors that need more focus in future studies.

In addition, an SDN component allows the self provision of connectivity services between hosts in our SDN framework. Moreover, when this block receives information about problems in the substrate network, it performs a reoptimization to overcome them in order to keep the good operation of the substrate network and all of the connectivity services.

A journal paper was already submitted to Special Issue on Communications and Networking in the Cloud, Computer Networks (Elsevier editor), which study the mapping approach in cloud networking. It compares the heuristic algorithm and link reoptimization algorithm proposed in this Dissertation with a different Integer Linear Programming formulation algorithm. A publication is now being prepared with respect to the SDN framework.

1.4. Dissertation Outline

The outline of this Dissertation is the following. In chapter 2 it is presented an overview of the concepts and technologies related with the work of this Dissertation. Concretely, it is presented the main platforms and commercial offers to CC, network virtualization and SDNs. Finally, it is presented related work in server, network and cloud networking mapping algorithms.

The proposed mapping algorithm is presented in chapter 3. It has a different goal in comparison with the base mapping algorithm presented in the state of art in order to maximize the gains for the operator. Associated with the previous chapter, chapter 4 presents the developed link reoptimization algorithm. This chapter analyzes different approaches to reach the best possible outcome remapping only the virtual links.

In chapter 5 it is described a functional component that fits in our SDN framework, the mapping and decision function, that aims to support connectivity services are supported. Finally, chapter 6 summarizes the work developed and presents the future work.

2. State of the Art

2.1. *Introduction*

In this chapter we introduce the themes related to the work developed in the scope of this Dissertation. Section 2.2 provides an overview of the cloud computing concept. The main service types and models are presented along with some of the most relevant platforms and commercial offers. In section 2.3 we look to networking concepts such as networking virtualization and SDN. We look to the fundamentals of these concepts as well as related platforms and commercial offers. Section 2.4 shows what we consider to be some of the most relevant related work in the area of resource mapping algorithms. This section starts by addressing algorithms that look to cloud resources and network resources individually. Further we present algorithms that tackle cloud and network resources in an integrated way.

2.2. *Cloud Computing*

2.2.1. *Concept Overview*

Cloud Computing (CC) is a new paradigm that has per basis old concepts (can be seen as the concatenation of those concepts), such as:

- ***Grid computing*** – is the technology that uses the capability of different computers to achieve a common goal (parallel computing).
- ***Utility computing*** – is the concept of pay for use of computing resources (i.e. computation, storage, services). This represents a big advantage because it allows the maximization of the resource utilization and the reduction of their operation costs.
- ***Autonomic computing*** – aims to reduce the management complexity of the computing systems by allowing self-management systems to work without human interaction. Related with this, CC wants mainly to reduce the resources costs automatically without focusing on the management complexity.

- **Virtualization** – forms the foundation of CC by giving the capacity of reaching the pool of resources from clusters of servers to applications on-demand. The virtualized server that provides the virtual resources for high-level applications is called Virtual Machine (VM).

However, it is hard to restrain the CC to a single definition. It seems that every expert or provider has its own definition, so we highlight two of the most cited definitions:

- “Cloud Computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction”. National Institute of Standards and Technology (NIST) [1] .
- “Cloud computing as a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies”. Gartner [2].

The main features of CC are:

- **On-demand** – allows the consumers to provision automatically computing capabilities. The services must be always available to access, allowing to consumers to modify their needs without human interaction with the provider.
- **Scalable and Elastic** – this is a key feature of cloud computing. The computer resources are easily expandable: they can be quickly freed and obtained in order to answer to the demand, most of the time almost automatically.
- **Pay-per-use / measured service** – cloud computing services are charged by the time of use such as the electricity bill, in terms of time (hours, days), data transfers or other use-based attributes delivered. Therefore, the client’s only pays for the amount of resources contracted, independently on the cost of the equipment. With CC the capital expenses (CAPEX) are converted into operations expenses (OPEX).
- **Easy access** – the internet is the environment used to connect with the cloud, which makes it accessible from every platform with internet connection (i.e. tablets, Personal Digital Assistant (PDA), laptops and mobile phones) and from almost every part of the world.

2.2.2. Services

CC has different services that attract different groups of consumers according to the different business areas. The Figure 2.1 shows the main services categories. More recently new services were added as the case of the Network as a Service (NaaS). Each service will be explained below:

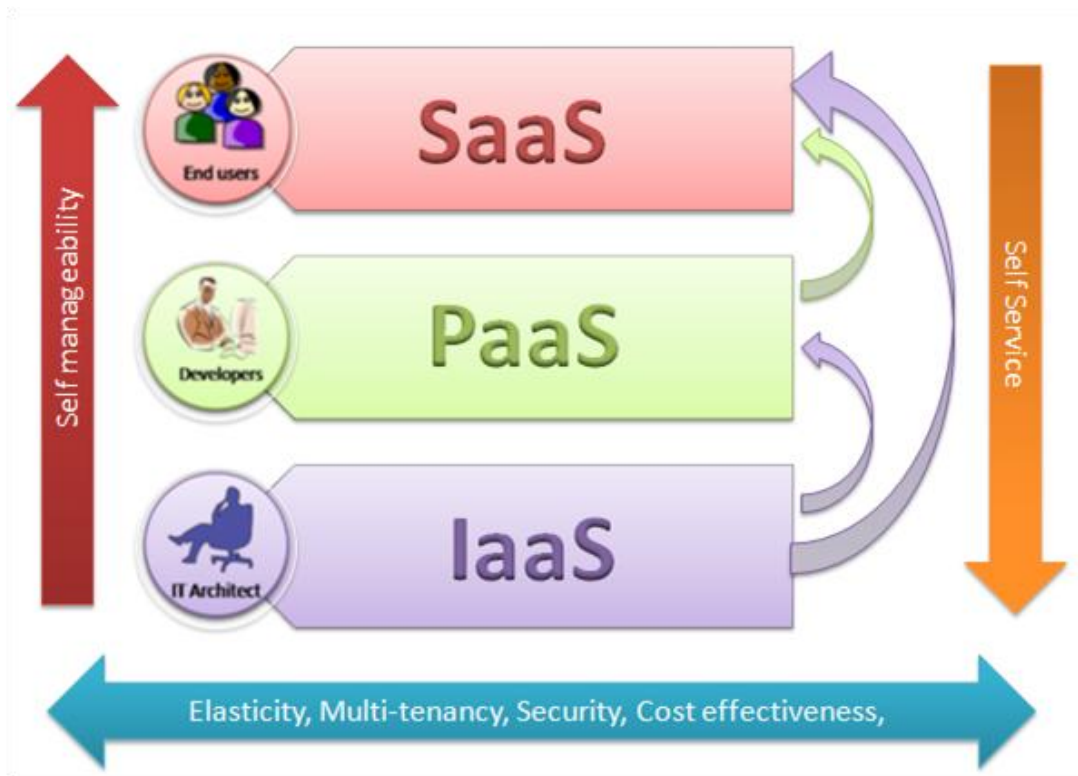


Figure 2.1 – Cloud computing services [3].

- **Infrastructure as a Service (IaaS)** – the lowest layer of the service category. It is used to support the remaining services. *IaaS* allows consumers to use infrastructure resources as a service. This service can be divided in sub-services as Computing as a service in which delivers computing power and VMs instances, Storage as a service if the consumer is only interesting in the storage of information managing the access to other users, and Database as a service if the consumer is interested in access and manipulate data through a database management system. These infrastructure services use software Application Programming Interface (API) to be available to access. The *Amazon Elastic Compute Cloud (Amazon EC2)* [4] and *GoGrid* [5] are examples of *IaaS* providers.
- **Platform as a Service (PaaS)** – is directed to software/application developers. It provides a programming environment for the developers to create, deploy, manage and test their applications. The *PaaS* consumers do not control nor “see” the underlying cloud infrastructure used to host their applications, but have control over the deployment of their applications and possibly of some configuration settings for the application-hosting environment. *Google App Engine* [6] and the *Microsoft Windows Azure* [7] are examples of *PaaS* providers.
- **Software as a Service (SaaS)** – the highest of the cloud service category. *SaaS* provides applications/software to end-users over the *Internet*. The control of the underlying layers (Infrastructure and Platform) is completely assumed by the infrastructure provider. The *Salesforce* [8] and the *Rackspace* [9] are examples of *SaaS* providers.

- **Network as a Service (NaaS)** – this service provides to the consumers the possibility to use the network/transportation connectivity of the provider infrastructure as a service. This allows current CC offers to interact directly and in a secure way with the network infrastructure. *NaaS* uses tenants, which allow the deployment of custom routing and multicast protocols (i.e. to make a better use of the network infrastructure in the consumer and the operator point of view). The common *NaaS* services comprise flexible and extended Virtual Private Networks (VPN) and Bandwidth on Demand (BoD). The *Pertino* [10] is an example of *NaaS* provider.

The providers mentioned in the examples of each service are presented in the sub-section 2.2.4.

2.2.3. Models

Cloud services can be deployed according to different models. The most common models are the public, private, hybrid and community. Each model is explained below:

- **Public Cloud** – is a model where the provider exposes the resources (i.e. application and storage) for the general public over the *Internet*. Within the same cloud, different resources belonging to different consumers can be hosted. In this case, the cloud provider controls and manages the cloud infrastructure and its resources.
- **Private Cloud** – this deployed model is more turned to a specific consumer that owns and controls the entire *private* cloud. The customer can contract a third party company to build and install it. There is an alternative to this model called *virtual private* cloud that allows having a *private* cloud environment within the infrastructure of a *public* cloud. In this case, the consumer is assured that all the data is stored and processed in dedicate servers, without being shared with other users.
- **Hybrid Cloud** – this deployed model combines both the *public* and *private* models. It allows the owner of a *private* cloud to expand its environment into a *public* cloud. This can be of extreme usefulness when the *private* cloud does not have enough resources to respond to the required demand.
- **Community Cloud** – probably this is the less explored cloud deployment model, it is based on a shared environment where different entities share their resources to create a single cloud environment. This model can be seen as a *private* cloud to a group of consumers instead of to only one.

2.2.4. Commercial offers and providers

This section gives an overview of some of the most known cloud offers in the market according to the different types of services presented in the sub-section 2.2.2. Most of the following providers have more services than the presented below.

Amazon Web Services (AWS) [11] was founded in 2006 and is one of the most popular providers that offer IaaS, PaaS and SaaS. AWS has more than two dozens of data centers spread among four continents. Moreover, it allows customers to specify the location (i.e. data center) of his resources. Some examples of specific services are explained below:

- **Amazon Elastic Compute Cloud (Amazon EC2 [4])** – this is an IaaS service where the consumer creates and manages VMs. The benefits are the velocity of VM server creation that allows to consumers rapidly increase or decrease the capacity required and the consumer only pays for what he really uses. The user is charged taking into account the type of VM (number of Control Processor Units (CPUs), memory, disk) chosen and the amount of data transferred from and to the VM, whether it is traffic circulating within the data center or to/from the outside.
- **Amazon Simple Storage Service (Amazon S3)** – this is an IaaS product, more concretely a storage as a service where AWS offers scalable storage services (raw Hard Drive Disk memory, HDD) for consumers to save data. The information can be accessed by VMs within the datacenters or by remote clients using the *Internet*. The prices consider the amount of reserved HDD, the amount of solicitations and the data transferred from and to the block storage.
- **Amazon Virtual Private Cloud (Amazon VPC)** – this allows the consumers to create and configure a virtual private network environment (i.e. private Internet Protocols (IP), private sub-networks, gateways) where cloud resources are connected to. The user is able to reach its resources via an IPsec VPN or through an operator-based VPN. In the latter case, the user has to use another service, the Amazon Direct Connect.

GoGrid [5], as an IaaS provider, gives computing (i.e. VMs) and storage as a service. The GoGrid has the particularity of providing virtual and physical infrastructure on-demand (i.e. servers, storage, and networking). It has available the models of *public*, *private* and *hybrid* clouds.

Google App Engine [6], as a PaaS provider, gives the necessary means for developers to use the best of this service in order to create, manage and deploy their applications as toolkits and programming languages as *Java*, *Python*, *Golang* (GO) and *Hypertext Preprocessor* (PHP). The last two languages are in an experimental stage at the moment. This provider, together with the *Salesforce*, created a set of libraries, web service API, with the objective to link their services.

Microsoft Windows Azure [7], as a PaaS provider, allows the consumer to host or to use web applications on the *Microsoft* datacenters. In common with the others providers, it gives to the consumer the capability of creating and managing VM images. *Microsoft Windows Azure* also gives applications to apply to the cloud as database banks and management applications.

Moreover, Software Development Kits (SDK) and programming languages (C#, Visual Basic, C++) are supported.

Salesforce [8], as a *SaaS* provider, gives the *Customer Relationship Management (CRM)* service. This service entails all the aspects of interaction between the company and its clients. This product was more recently divided into several categories as *Sales Cloud*, *Service Cloud*, *Data Cloud*, *Collaboration Cloud* and *Custom Cloud*.

Rackspace [9] is a complete provider specialist of infrastructure, email and applications hosting services. The main service that *Rackspace* provides is the managed support that is an on-demand support where proactive services are provided. It also provides another service called fanatical support. This service is more complete, which tries to resolve every kind of problems that consumers of cloud infrastructure have. A difference from *Amazon* is that *Rackspace* does not charge the data transfer between their clouds.

Pertino [10] is a *NaaS* provider that introduced a service called the “Cloud Network Engine”. This service is the combination of the SDN technology with the Wide-Area Network (WAN) virtualization that allows consumers to create a dynamic virtual private network environment in the cloud. For example, it allows an organization that has multiple offices to create a virtual private network in the cloud connecting all the offices, using invitations to join partners to their network.

2.2.5. Cloud Management Platforms

Cloud management platforms are the foundation of cloud offers. In this section we give an overview on some of the most known and open cloud management platforms. The common characteristics of these management platforms are the incorporation of self-service interfaces, providing some level of optimization and having capability to perform metering and billing of resources. *OpenStack*, *OpenNebula* and the *CloudStack* are the platforms that we will talk about.

- **OpenStack [12]** – was created by an initiative that joined *Rackspace* and *NASA*, with the objective to create a cloud computing offer able to run on standard hardware. Nowadays it is managed by the *OpenStack Foundation*. *OpenStack* is constituted by individual modules (each with different APIs), which provide different functionalities/services. The official module are: the Identity module (codename *Keystone*) which is responsible for authentication and authorization; the compute module (codename *Nova*) responsible for offering computing as a service – similar to *Amazon EC2*; the storage module (codename *Swift*) responsible for providing storage as a service – similar do *Amazon S3*; the image module (codename *Glance*) that allows the hosting VM images and operating systems; the block storage module (codename *Cinder*) responsible for managing the storage blocks used by the computing service; and the network module (codename *Quantum*) that provides network as a service (to create and manage routing elements and sub-networks). Note that many other modules/services are under development. The key factors of this platform are its level of control and flexibility due to its modular and open source nature,

being an industry standard platform used by many major companies in the industry [13]. In Figure 2.2 it is noticeable how the interactions are done by the *OpenStack* software. It is important to highlight the high involvement of industry in the development of this platform [14], i.e. Canonical, HP, IBM, AT&T, Red Hat, Cisco, Ericsson, NEC, VMware and many others.

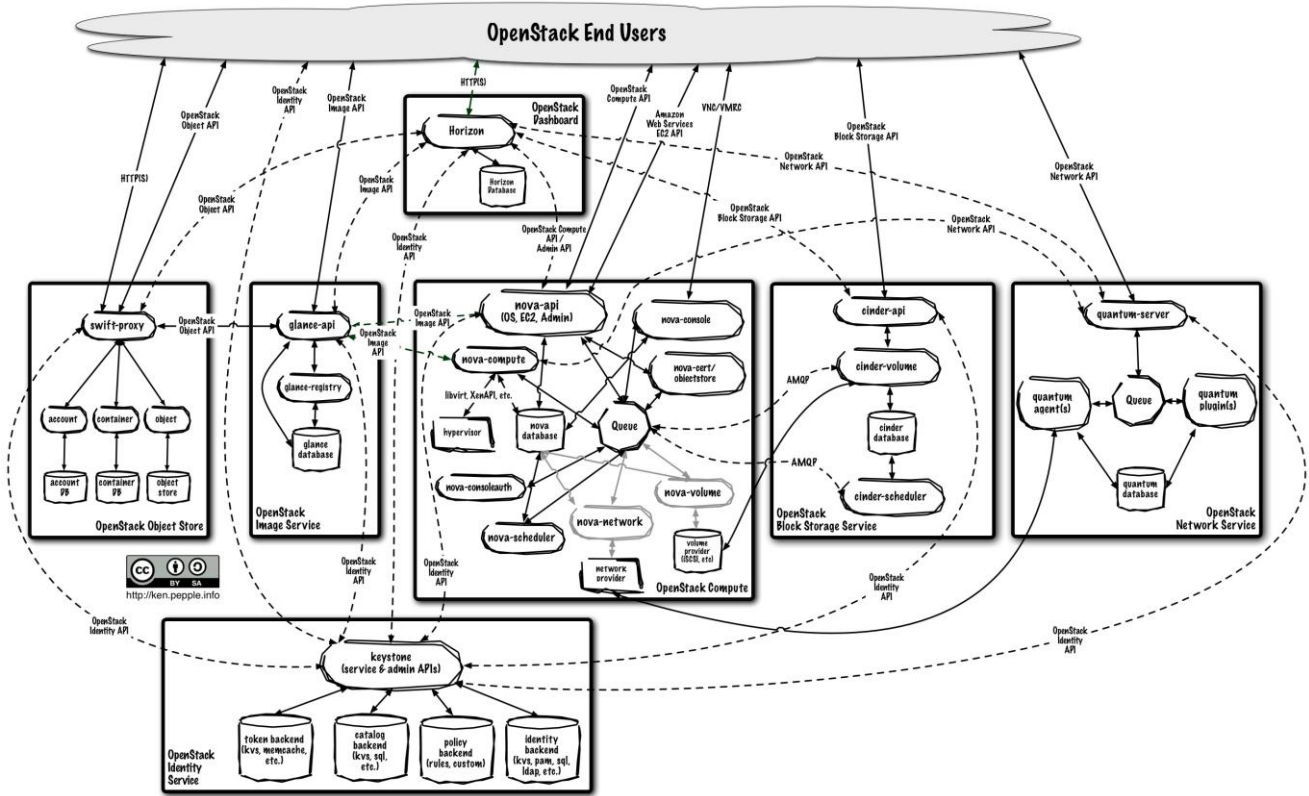


Figure 2.2 – *OpenStack* architecture [15].

- CloudStack [16]** – was a private source created by VMOPs and changed to *Cloud.com*. After that change, the core of the *CloudStack* was set as free software until the *Citrix* involvement. After that, the entire *CloudStack* started to be open source and being managed by the *Apache Software Foundation*. With that involvement, *Citrix* ceased its involvement with *OpenStack* and adopted the *CloudStack* to implement in its clouds. Some features of *CloudStack* are the easiness of deployment and the availability of a dashboard, Graphic User Interface (GUI), for management of zones. The zones are constituted for a set of clusters (known as pod) connected by layer 2 switches. A cluster is constituted by a set of hosts managed by the same hypervisor. The *CloudStack* supports hypervisor, as *VMware* via *vCenter* [17], *KVM* [18] and *XenServer* [19]. The host is the basic unit of scale. In figure Figure 2.3 it is shown the network setup in a zone with all its elements. Furthermore, *CloudStack* has a replete set of RESTful APIs available and also has APIs compatible with the *Amazon* services (*AWS EC2* and *S3*). It is important to mention that *CloudStack* supports SDN. Some of the leading telecommunications companies have chosen *Apache CloudStack* to power their cloud services, as *BT*, *China Telecom*, *EVRY*, *IDCFrontier*, *KDDI*, *KT*, *NTT Communications* and *Slovak Telecom* [20].

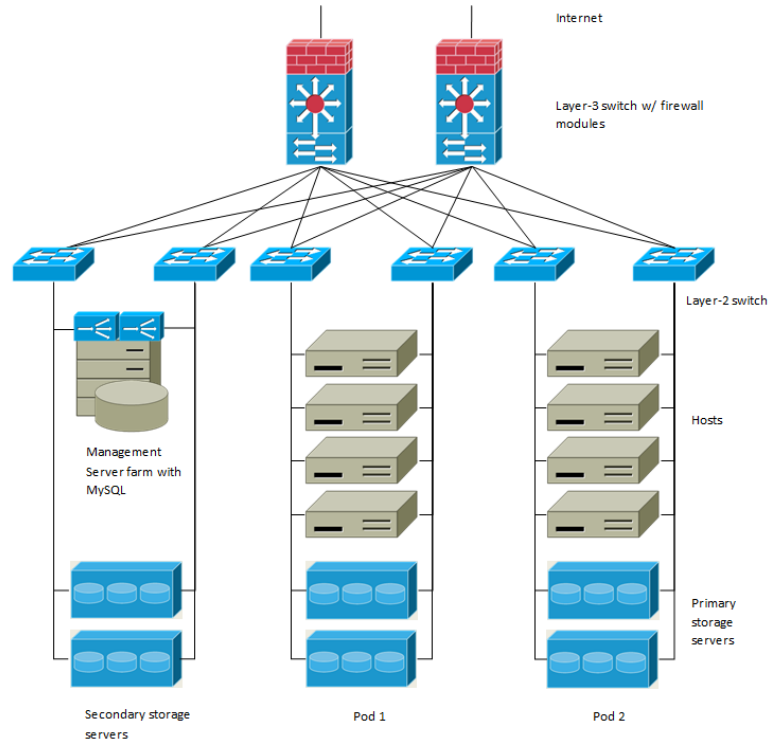


Figure 2.3 – “Networking Setup in a zone” [21]

- **OpenNebula [22]** – started to be an individual research project to become an open source project developing a standard solution for building and managing virtualized enterprise data centers and *private* clouds. *OpenNebula* is modular to allow its integration with tools and services in the virtualization, cloud environment and datacenter management. Its primary use is to manage a *private* cloud inside data centers or inside clusters. *OpenNebula* also supports *public* clouds, by providing cloud interfaces in order to expose its functionalities for virtual machine, storage and network management. It is used to manage private clouds and manage data centers virtualization by many leading organizations as *IBM*, *DELL* and *SAP* [23]. Nowadays it is in the 4.0 version.

In this sub-section we have highlighted the infrastructure cloud management platforms due to the close relation to the scope of this work. However, it is important to highlight that there are also available platforms at the *PaaS* level, such as *OpenShift* and *CloudFoundry*.

2.2.6. The Network's role

The network is an important part of the cloud due to the fact that it is the way to access it. The network has responsibilities in the delivery and performance of the cloud services, and because of that, it becomes a strategic asset. Moreover, the network is also seen as a resource to be provided as a service (i.e. as a cloud service itself). The following reasons justify that importance [24]:

- **Secure and manage the cloud** – the network is the only IT asset that contacts each other IT resource. Thereby, a secure and manageable network means a more secure and better cloud services experience.
- **Cost-effective delivery platform** – nowadays the network is the easiest way to connect every user from different places with the cloud environments, using different types of devices (i.e. laptops, mobile phones and tablets). In order to improve the delivery of applications for all the mentioned types and places, it is possible to push the applications to the network, and from there, to serve every end-user.
- **All time accessible** – currently the coverage of the network is done by a set of different technologies (i.e. 4G coverage, home networks, corporate networks, public hot spots and Wi-Fi) that allow an almost all access.
- **Virtualization** – with the virtualization it is possible to host the applications, computer and network resources in most places. The network is used to connect the different components from the different locations allowing better cloud computing services in every level (i.e. providers' competition, consumer and provider services performance).

2.2.7. "Cloud Networking"

Nowadays the assurance of security, reliability and performance for the cloud services are achieved mainly in the enterprise sector with the use of Operator-managed VPN service models (Border Gateway Protocol (BGP), Multiprotocol Label Switching (MPLS) [25], Internet Protocol VPN, Virtual Private LAN Service (VPLS) [26],[27]). The main disadvantage on those services is the lack of elasticity and self-provision (i.e. some of the main characteristics of cloud computing). Furthermore, the business relation between consumers and cloud providers is done by the use of Service Level Agreement (SLA) in one of two possible ways: having that SLA with the cloud provider and use the *Internet* as the way of accessing the service; having two SLAs, one with the cloud provider and another with the network provider that give the connection between the consumer and the hosting cloud. Taking into account the referred aspects, it is noticeable the need to integrate network and cloud resources together. Figure 2.4 shows the foreseen evolutionary process of cloud and network.

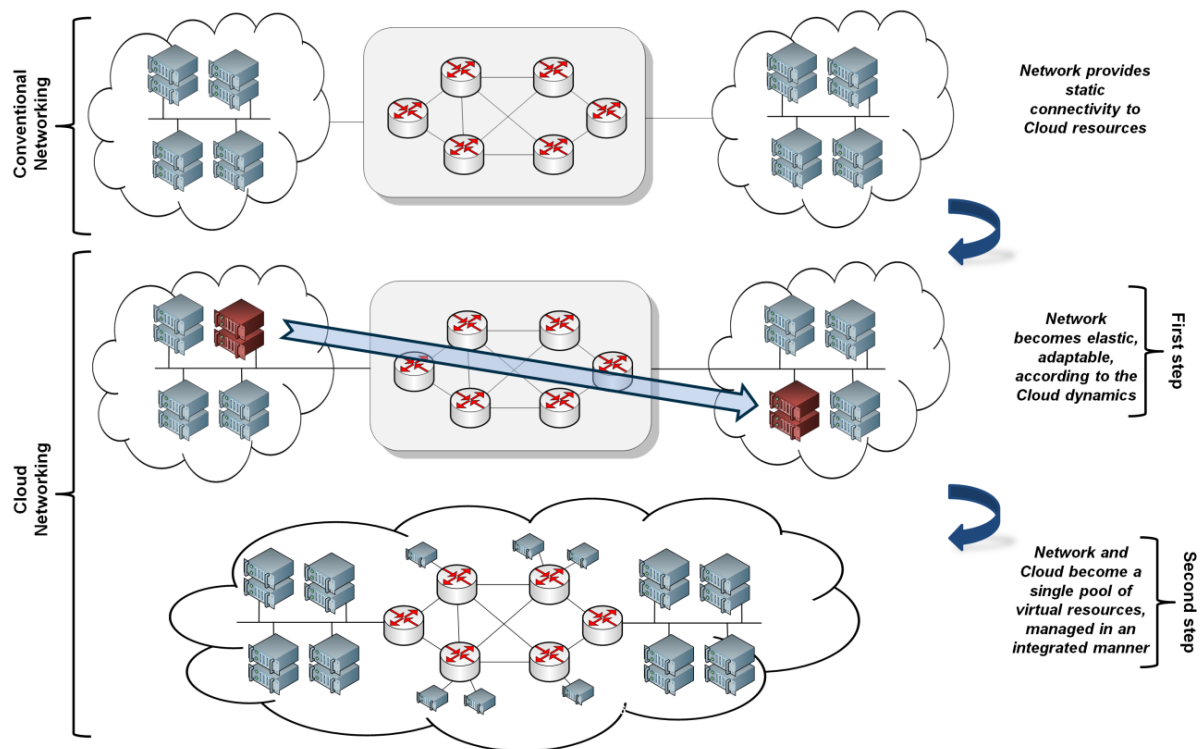


Figure 2.4 – The evolutionary process of cloud and network [28]

2.3. Networking

2.3.1. Introduction

Networking is the ability to connect elements with the objective of sharing information. Related with CC, we will present some networking considerations to the public and private models [29]. For public clouds, the *Internet* is used as the basic networking platform for consumers to connect to the cloud, allowing a broad set of access methods and connectivity technologies. The providers that use this platform expand their security boundaries to the *Internet* and beyond, because of that the companies need to enforce the security and privacy policies covering the public domain. Another way to have access to the cloud is by extranet. This is not so problematic in security point of view (i.e. leveraging firewall and encryption technologies). With extranets, the network performance can be improved with the use of symmetric WAN optimization technologies. However, extranets can present problems cause by its organization. With that the cost and performance can be seriously affected. In the cases of private clouds, the network is controlled by the contracting company. In these cases the company needs to evaluate the reliability, performance, security and expanse of the network that will support a cloud-based environment, once it is their responsibility the overall network infrastructure and management.

Initially the networks were designed to be static, inflexible, isolated and managed separately from other data center operations. However, to deliver the benefits of cloud, applications, servers and storage, the network requires a new approach, which all those elements need to be

considered as a system, managed and provisioned together. The network designed should be simplified, with the standardization of devices and protocols, modular to enable easily scalability, the virtualization must be applied to minimize the number of physical equipments, specific features of the providers should be minimize the most possible to allow simpler troubleshooting and management. In the following sub-sections we will present some initiatives and technology that try to overcome these problems/fragilities.

2.3.2. Network Virtualization

2.3.2.1. Concept Overview

Network virtualization is the emulation of several computing networks (hardware and software resources) in the same substrate network, Figure 2.5. From the user point of view, the Virtual Network (VN) has the same behavior as a physical network. For the operator, network virtualization represents a great way to monetize with their unused resources, run network protocols independently of the physical network and allow their adjustment to the services that are deployed on top of it.

Moreover, network virtualization allows multiple heterogeneous network architectures to cohabite in the same physical network, sharing the same physical resources. It also provides flexibility, security and increased manageability.

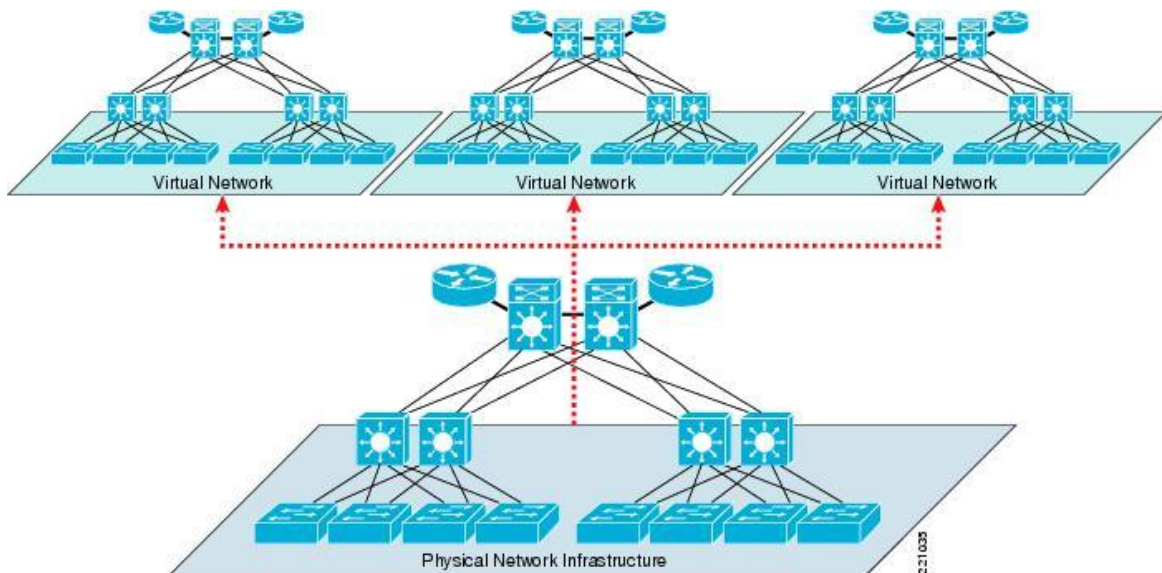


Figure 2.5 – Creation of Virtual Networks [30]

2.3.2.2. Architecture

The basic entity of the network virtualization is the VN. It is composed by nodes that share links between themselves, forming a virtual topology. Every virtual node is hosted in a particular physical node. The virtual links are hosted in the physical connections between those physical

nodes. The hosting process takes a portion of the network resources. The consumer of the VN is free to implement any desirable mechanisms, services, and protocols.

Main principles [31]:

- **Coexistence** – is allowed the hosting of multiple VNs in the same physical network. It is possible to host partial VNs in different physical infrastructures belonging to different providers.
- **Recursion and Inheritance (nesting)** – VNs can spawn into other VNs creating a VN hierarchy with parent-child relationship. The older VN shares a part of its resources with the new VN. The architecture attributes and constraints of the parent VNs can be transferred to child VNs. This allows the providers to add value to the spawned child VN.
- **Revisitation** – this allows the use of a single physical node to host multiple virtual nodes from the same VN (i.e. virtual switch, virtual routers) in order to handle with functionalities, rearrange its network structure and simplify the management of a VN.

2.3.2.3. Platforms & Commercial offers

In this section we will present some existing network virtualization platforms and commercial offers.

- **GENI [32]** – *Global Environment for Network Innovation (GENI)* is a virtual laboratory initiative that provides realistic experimental facilities (customized virtual network testbeds). The goal is to evaluate alternative architectural structures (deploying prototype networks and run experiments). Some advantages of the *GENI* are: can be programmable at any level of abstraction (i.e. optical, IP); the node behavior can be controlled; and it is allowed the incorporation of network technologies such as sensors, wireless and optical.
- **CABO [33]** – *Concurrent Architectures are Better than One (CABO)*. Similar to *GENI*, it also can be programmable at any level of abstraction, but it provides a separation between the physical network infrastructure and the application that runs over it. Consequently, this transfers the responsibility for the physical devices to the infrastructure providers. The service providers can run different end-to-end services concurrently, and the infrastructure providers can compete to give better services, hence the services can be attached to different infrastructure providers. The *CABO* goal is to provide a common framework to improve the network services, and simultaneously provide better management operations.
- **4WARD [34]** – a former European Project focused in virtualization, discovery, monitoring, management and provisioning technique for network resources. *4WARD* goal was to make the development of network and networked applications the most easily and faster

in order to advance an affordable communication services. They worked on an overall framework that allows the coexistence, inter-operability and complementarity of several networks.

- **NICIRA [35]** – gives an intelligent abstraction layer between network and end-users. By the management of a distributed control system (network components and connections), it is possible to transform the network into a pool of network resources with the objective to be shared by a higher number of isolated VNs. It uses Open Virtual Switch (OVS) to create a software abstraction layer between the physical network and the servers. The control system exposes a RESTful web services API.

2.3.3. Software Defined Networking

2.3.3.1. Concept Overview

In the past, the network elements (network applications, custom control plane and custom silicon) were integrated all together, in a vertical way. That reality was becoming obsolete and a new research was initiated to overcome the limitations of that design. Nowadays, the SDN approach is to build computer networking equipment to separate the abstract elements of those systems. Basically SDN brings to network infrastructure the kind of programmability that is common in the computer world. The control is decoupled from the hardware (i.e. router and switch) and given to the controller (i.e. software application). In general this can be seen as the same abstraction level that CC does with the servers.

The benefits of SDN are clearer in cloud environments, where VMs are added and moved constantly, which creates problems in terms of adaptation and scalability. With SDN, the control can rapidly reprogram the data plane elements to face the new demand, and the management of the network becomes much easy to effectuate in a large scale.

2.3.3.2. Architecture

The SDN main architecture proprieties are: it is modular; programmable (APIs have their structure well defined to perform exchange of data); economical; agile; and can operate in the layer 2 (Ethernet switches), layer 3 (Internet routers), layer 4 (transport switching) or application layer switching and routing. Figure 2.6 shows the planes that compose the SDN architecture.

The interaction between the application layer (application plane) and the control layer (control plane) is performed by APIs. The controller receives the information from the applications and translates it into configurations to apply in the substrate elements. Thus, these applications only see an API of the controller. The controller communicates with the substrate elements through the *OpenFlow* protocol. The substrate elements will simply be coordinated by the controller instead of performing decisions, as it was done before in past networks (i.e. if a new packet

arrives to the switches and there are no rules for it, the packet is forwarded to the controller, which will then decide what to do with it).

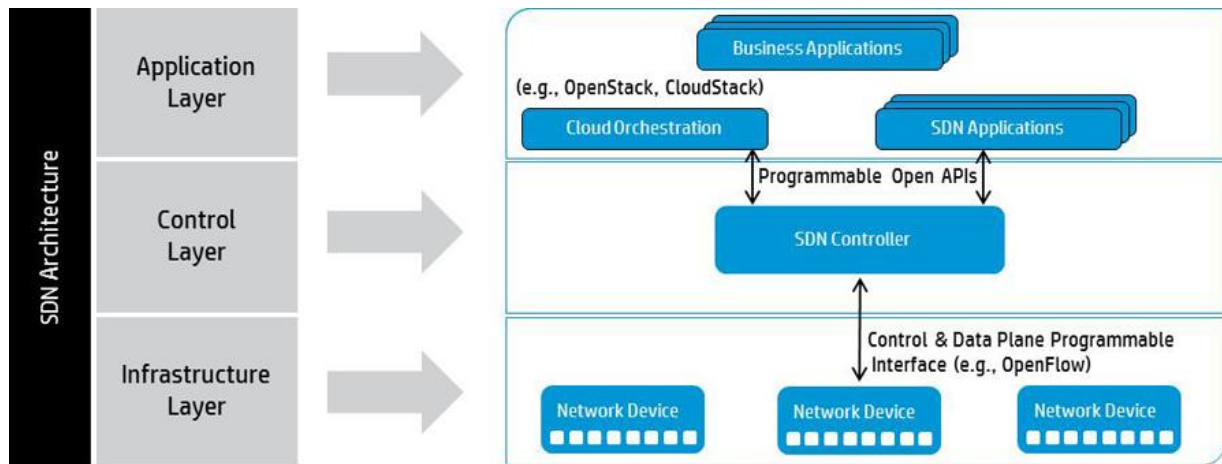


Figure 2.6 – SDN architecture [36]

2.3.3.3. *OpenFlow*

OpenFlow is a communications protocol that accesses and manages the forwarding elements (switches and routers) of the data plane. The controller configures the forwarding elements in its decision level (called flow-tables) through the *OpenFlow* protocol. Hereby, the functions of the routers and switches are simplified, because now those elements only need to identify the packets that are arriving and match them with the rules presented in their internal flow-table. In the cases the packets are new (the element does not have any rules for those packets), the element sends them to the controller, for the controller to decide what will be done with those packets in the future (i.e. drop all or specify a port to do the forwarding).

OpenFlow is added as a feature to the commercial switches, routers and wireless access points, to allow researchers to run experiences without accessing the internal core of the network. With the separation of the data plane from the control plane, it is possible to use more effectively the resources of the network. Related with CC, this can resolve the problems of scalability and mobility of the VMs and its consequences for the network (reliability and management).

Several large companies already offer switches that support the *OpenFlow* protocol as *HP*, *NEC*, *IBM* and *Juniper*. We present some commercial offers in the sub-section 2.3.3.5.

2.3.3.4. *Controller Platforms*

The controller platforms are the operations systems of SDN technology. There are several offers of controllers that use the *OpenFlow* standard. Following, we present some of the most relevant *OpenFlow* controller platforms.

- **NOX [37]** – is the first *OpenFlow* controller to appear. It provides a high-level programmatic interface upon which network control applications can be built. *NOX* also allows a centralized programming model for an entire network (whether large or small networks). With *NOX*, developers can control all the network connectivity as forwarding and routing. The current release includes full policy engine, central management for the switches and user admission control. Application can be written in C/C++ or python programming languages. There are some controllers based on *NOX*, such as *POX* [38] (python-based), *Jaxon* [39] (java-based) and *Simple Network Access Control (SNAC)* [40] (uses a web-based policy manager to manage the network).
- **Floodlight [41]** – is an open source controller that works with physical and virtual switches through *OpenFlow* protocol. Its main characteristics are: easiness of use; open community for developers; tested and supported. *Floodlight* is designed to be a high-performance controller and is supported within the *OpenStack* cloud platform as part of the networking module (codename *Quantum*).
- **Ryu [42]** – is a controller that supports the latest version of *OpenFlow*, version 1.3 and *Nicira* Extensions. Just like *Floodlight*, *Ryu* is supported by *OpenStack* and also works with physical and virtual switches.

Those controller platforms were just some examples of the controllers available nowadays. In our SDN framework, it is used the *Floodlight* controller because it allows the interaction with OVS, the forwarding elements of our network.

2.3.3.5. Commercial offers

At the moment the available commercial offers are mostly in the segment of hardware equipment, (i.e. switches with *OpenFlow* support). Table 2.1 provides a list of the manufactures that sell these equipments, the equipments model and the supported *OpenFlow* protocol version [43]:

Manufacturer	Switch Model	Version
Hewlett-Packard	8200zl, 6600, 6200zl, 5400zl, and 3500/3500yl	v1.0
Brocade	NetIron CES 2000 Series	v1.0
IBM	RackSwitch G8264	v1.0
NEC	PF5240 PF5820	v1.0
Pronto	3290 and 3780	v1.0
Juniper	Junos MX-Series	v1.0
Pica8	P-3290, P-3295, P-3780 and P-3920	v1.2

Table 2.1 – Commercial switches that support the *OpenFlow* protocol

2.4. Resource Management Algorithms

Associated with CC and network virtualization, there are several challenges related to resource management, such as resource mapping, resource optimization/reoptimization. In this section, we provide an overview of some of the most relevant state-of-the-art works in this area: management of cloud resources (namely VMs) and the network virtualization area. Finally, there is a sub-section dedicated to the integrated management of cloud and network resources, which is the base for the work developed in this Dissertation.

2.4.1. Cloud

Virtualization allows the sharing of a physical resource by a set of VMs. This paved the way for the emergence of cloud computing as mentioned in the section 2.2. From a provider's point of view, there is the need to optimize the initial allocation of VMs and to reoptimize for the providers benefit (i.e. gather all VMs in a reduced set of physical hosts in order to reduce energy consumption within a data center), and also to fulfill the VM's requirements (i.e. need at a certain point in time to move a certain VM closer to the end-user). These are extremely important issues and below we present work in the area.

Bouyoucef *et al.* [44] proposed an optimal algorithm approach that performs the allocation of VMs into data centers, from a pool of data centers. The authors aim to minimize the latency between the data centers and the consumers. Moreover, they aim to minimize the Round-Trip-Times (RTTs) between the data centers and the groups of users. The approach has two different behaviors, before reaching the substrate saturation, in which the utilization of the substrate increases directly with the increase of groups of users without performing reoptimization (nodes and links); after reaching the substrate saturation, the algorithm shifts and performs reallocation of virtual servers.

Csorba *et al.* [45] proposed decentralized, dynamic and self-organizing techniques to allocate VMs in public and private clouds. The authors aim to improve the scalability properties and the hosting re-allocation issues using intelligent agents that are responsible for physical server discovery and allocation decisions. The proposed techniques are able to react to the change of load in the physical servers and resist to severe failures of the physical infrastructure. It is important to mention that the mapping of VMs into physical servers is performed by near-optimal heuristics.

Moreover, Ma *et al.* [46] proposed a new model using the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) method to achieve better load balancing of the VMs in large-scale cloud computing environment, performing less VM migrations in comparison with optimal solutions. The authors try to avoid wasting resources as a result of under-utilization, or avoid lengthy response times due to over-utilization.

2.4.2. Network

One of the greatest challenges that virtual network management brings lays on the fact that it needs to deal with two types of resources at the same time, nodes and links. From a mapping problem point of view, this is considered an NP-hard problem. Below it is presented some work with respect to the virtual network mapping.

Zhu and Ammar *et al.* [47] address the VN mapping problem by proposing a heuristic algorithm. The authors let the algorithm to reoptimize when a new network requests arrives in order to achieve better performances. More than one approach to reach the best optimization performance was presented. The goal is to maintain a load balance in the physical substrate, using the concept of node stress (number of virtual nodes running on the physical node) and link stress (number of virtual link running in the physical links) to determine the load in the substrate. After this, the algorithm calculates the Neighborhood Resource Availability (NR) for each node, taking in consideration both stresses, in order to decide from which physical node will start the allocation. The physical node that presents higher value of NR will be selected to be the starting candidate to host the initial virtual node of the upcoming virtual network request.

Nogueira *et al.* [48] use an approach based in [47], but considers the heterogeneous hardware features of the physical nodes, i.e. CPU load, CPU frequency and memory. With those parameters the authors aim to perform a better load balance of the physical substrate occupation. It uses those parameters to calculate the node stress (occupation levels of the hardware features and VMs hosted) and the link stress (occupation levels of the substrate connection), in order to evaluate the potential (is the inverse of the multiplication of the node stress with the links cost, that is the link stress between the physical node in evaluation with the candidates of its virtual nodes neighbors) of the physical nodes. The physical nodes that present higher potential will be chosen to host the virtual nodes from the virtual networks requests.

Chowdhury *et al.* [49] separate the link mapping problem and the node allocations problem into two different stages. For the links mapping, the authors propose a multi-commodity flow algorithm, and for the node mapping they propose an integer program (deterministic rounding techniques and randomized rounding techniques). The authors perform an analysis in terms of acceptance ratio, revenue and resources usage (links and nodes), which are the most interesting characteristics to operators by providing the best way to get higher profit. In [50] the authors combine the mapping of the nodes and links simultaneously with use of a backtracking method based on subgraph isomorphism. With this mechanism, if a mapping decision fails, it goes back until the previous valid mapping decision, and changes the flow of events in order to avoid costly remapping. This proposition considers the online version of the network mapping problem.

Furthermore, Melo *et al.* [51] present an integer linear programming formulation to achieve an optimal mapping of virtual network resources, which considers resource management aspects. The author compares the performance of the optimal formulation with the heuristic work proposed in [48]. Moreover, Yu *et al.* [52] proposes a mapping algorithm that enables path splitting and link migration during the embedding process. However, this process can lead to a level of fragmentation that is unfeasible to manage on large scale networks.

Houidi *et al.* [53] propose exact and heuristic optimization algorithms for the provisioning of virtual networks involving multiple physical substrates. To split the virtual networks and provision into multi-providers infrastructures, [53] uses max-flow min-cut algorithms and linear programming techniques. This work is important because it does not limit the mapping of a virtual network to a single substrate, which can allow the usage of different domains to perform the hosting of the virtual network and use the characteristics of the different domains in terms of prizes, law and jurisdiction.

The above mentioned works provide different solutions for the VN mapping problem, however they lack a fundamental requirement to our work, the consideration of cloud resources.

2.4.3. Cloud and Network

In the past years, several works in cloud and network resource management have been developed. These works already consider both cloud and network resources together. Jiang *et al.* [54] analyses the interplay between traffic engineering, the network part of the problem that is under the responsibility of the Internet Service Provider (ISP), and content placement, the server part of the problem under the responsibility of the Content Provider (CP). The authors study 3 different models that differ in the amount of cooperation between the ISP and the CP, concluding that separating the traffic engineering and server selection leads to sub-optimal equilibrium. Moreover, the authors conclude that extra visibility might also result in a less efficient outcome.

Roy *et al.* [55] presents an empirical study of bin-packing heuristic algorithms to face the resource allocation problem for Distributed Real-time Embedded (DRE) systems. Li *et al.* [56] takes into account constraints, such as latency and bandwidth consumption, in the placing decision method, making it suitable to ISPs that already know the topology of its network to distribute server.

Enokino *et al.* [57] and [58] address the problem of the energy costs inside datacenters. In [57], the authors evaluate Power Consumption-Based (PCB) and Transmission Rate-Based (TRB) algorithms to select a server in order to reduce the total power consumption. The total power consumption is reduced by those algorithms in comparison with the traditional Round-Robin (RR) algorithm, and the authors conclude that PCB is more practical than TRB. In [58], the authors propose an Extended Power Consumption-Based (EPCB) algorithm that is an enhancement of PCB algorithm. With EPCB, the authors aim to reduce more the total power consumption.

Kantarci *et al.* [59], [60] and [61] present more studies in the cloud networking field. The work in [59] studies the delay minimization in the cloud network through a Mixed Integer Linear Programming (MILP) formulation. Furthermore, in [60] the authors address the reconfiguration of the cloud network with the objective of maximizing the energy savings and propose two heuristic approaches benchmarked by MILP approaches. In a later work, in [61] the authors study the trade-off between energy savings and delay minimization and propose a heuristic.

The previous works try to overcome challenges in the allocation problem from a network perspective, from a cloud perspective, and from an integrated way. Nonetheless, all the works do

not look to a deployment of cloud and network resources within a complete virtualization environment. Moreover, most of the works do not consider the virtualization of the network, or they take into consideration Quality of Service (QoS) constraints and do not try to optimize the use of network resources in order to favor the embedding of future requests. Furthermore, these studies do not take into account the interplay between particular cloud resources such as CPU capacity, memory, and storage.

The following works aim to tackle the points above mentioned. These works are related with the “cloud networking” view presented in sub-section 2.2.7. Notably, these works are relevant for this Dissertation because they represent the base algorithms of the chapter 3.

Romeu *et al.* [62] create a simulator and a mapping algorithm that already incorporates the cloud elements and network as a single pool of resources based on some of the principles of [48]. The mapping algorithm uses the potential of the physical nodes to perform the allocation decisions. The authors calculate the potential of the physical nodes through the analysis of its node stress (number of VMs hosted and their resources utilization) with the link cost (link stress between that physical node and the candidates of the virtual nodes neighbors). Furthermore, the authors propose new methods to calculate the node and server stress considering the hardware features of those elements. The objective is to balance the load occupation in the physical substrate. Soares *et al.* [63] present the study of non-proportional approach for the servers stress calculation and compare with the server stress calculation used in [62]. The authors also study the performance of the mapping algorithm in a real environment.

In this Dissertation, we will proceed the line of thought of the last two works presented in this section, in which the network consideration and optimization is very important in the placement decisions. With that in mind, we see an opportunity of research to improve the algorithm performance in order to obtain higher gains for the operator. Moreover, we want to study the optimization of resources occupation of the physical infrastructure in order to maximize those gains.

2.5. Summary

This chapter described the requirements that a solution has to present to be considered as a CC solution, followed by the current CC services, models and some commercial offers. Nowadays, there are available several types of CC services, all with a common aspect: the usage of a network. The network plays an important role in how the CC services reach their public. In addition, CC popularity is growing and evolving (i.e. appearing new services and business models) which introduce problems to the deployment of those services over the old concept beyond the original network (i.e. *Internet* uses the best-effort policy). Hence new approaches, such as network virtualization and SDN, have emerged in order to improve the network performance. This chapter presented an overview of both virtualization and SDN architectures, platforms and some existing commercial offers. Finally, it was presented a state of art in the resource mapping algorithms to work inside the cloud datacenters, in networks and in cloud networks.

The cloud computing is a new reality with opportunities to developers to contribute to their expansion. With this in mind, the first part of this Dissertation focuses in the resource mapping in cloud networking environment. Related with this, the later algorithms presented in the sub-chapter 2.4.3 have the objective to balance the load inside the provider infrastructure. However, that policy restricts the embedding of new requests penalizing the profits of the provider. In this Dissertation, we propose a cloud networking mapping algorithm that maximizes the profits of the operator by accepting more requests and use less resources of the provider infrastructure. Moreover, a link reoptimization algorithm is proposed and studied in order to analyze its benefits for the profit of the operator. The second part of this Dissertation takes the considerations achieved with the proposed mapping and link reoptimization algorithms, and proposes a module to close the initial cycle of our SDN framework. Once SDN is a recent approach, this module has the objective to receive and activate connectivity services. Additionally, this logical component has network management responsibilities related with those services.

3. Cloud Network mapping algorithm

3.1. Introduction

In the state of the art, section 2.4, we presented work in resource management algorithms for servers inside the cloud, networks and cloud networking. Related with cloud networking, in this chapter we aim to improve the embedding of Virtual Infrastructures by reinforcing and modifying the mapping algorithm in [62]. VNs are described as a set of routing nodes and links with its own characteristics, respectively. However, a VI is more than a VN, because it also considers servers nodes.

In section 3.2 we present a description of the improved mapping algorithm. We divide the mapping algorithm in four different parts: creation of the candidates list; calculation of the substrate stress; decision process; and link mapping.

In section 3.3 we argue that the list of candidates is not good enough in order to maximize the embedding of VIs. Moreover, the mapping algorithm was missing a routine to avoid the system to fail prematurely in the decision process. In order to narrow down the list of candidates, we implemented a first level try routine to remove pseudo-candidates, and a check routine to strengthen the success of the decision process.

Furthermore, in section 3.4 we argue that the elements that compose the potential of each candidate are poorly adjusted to maximize the embedding of VIs and the profits of the provider. With the aim to overtake that limitation, we present a new way to calculate the link stress where the utilization of links is penalized. Moreover, we assure that the node stress of the candidates is never zero.

In sections 3.3 and 3.4 it is performed individual performance tests to analyze the implications that each new modification brings in terms of acceptance ratio, bandwidth occupation, revenue ratio and cumulative usages of nodes and links for the same VIs arrival rate. The revenue of a VI depends on its resources (number of nodes, node characteristics, number of links, and characteristics of links) and its lifetime.

Finally, in section 3.5 we compare the final performance of our mapping algorithm with the base algorithm [62]. This evaluation considers the usage over time and different arrival rates of VIs requests. All these tests consider the online scenario of the mapping problem.

3.1.1. Discrete event simulator

We updated the simulator proposed by Monteiro [62], which simulates the environment of an operator physical infrastructure, where the VI requests arrive and depart. The simulator already distinguishes the physical infrastructure elements by routers and servers as proposed by [63]. The VI requests are formed by VMs with independent attributes of Hard Disk Drive (HDD), Random Access Memory (RAM), CPU frequency and type (nodes and servers), depicted in Table 3.1. The VMs of the router type can be only hosted in the physical routers, and the VMs of the server type can be only hosted in the physical servers. The VI requests can be accepted or refused depending on the conditions of the physical infrastructure in terms of free RAM and link bandwidth. To every VI request it is applied the mapping algorithm, which is responsible for the allocation decisions related with node hosting and link bandwidth reservation, updating the occupation levels of the physical infrastructure. In the successful cases, the mapping algorithm indicates which links are needed to be reserved, and which physical nodes are hosting the VMs of the VI request. This simulator registers the time that each mapping process takes, and gives as a result the averaged values of all the parameters related with the physical infrastructure resources, number of VM per node, HDD memory in use, Load in use, RAM in use among other information to further analysis. We added the capacity to obtain data over time related with the physical infrastructure parameters referenced before, and the information about the nodes and links utilizations, to be able to analyze the overall energy consumption (green study).

For each repetition, the simulator randomly creates a set of different physical infrastructures according to a pool of input parameters (Table 3.1), and randomly creates a set of different VIs requests to be mapped in those physical infrastructures with a correspondent lifetime and arrival interval. To compare the efficiency of each improvement applied, it is used the same VIs requests and the different physical infrastructures created for each run for every different measure. The focus of this chapter is to understand the global physical infrastructure implications for each change applied in the base mapping algorithm.

		Physical Networks	Virtual Networks
Router Nodes	N. CPUs	{2; 4; 6; 8}	{1; 2; 3; 4}
	CPU Freq(Hz)	{2.0-3.2 / 0.2 steps}	{2.0-3.2 / 0.1 steps}
	Memory	{2; 4; 6} (GB)	{64; 128; 256; 512} (MB)
Server Nodes	N. CPUs	{8; 16; 32; 64}	{1; 2; 4; 8; 16; 32; 64}
	Storage (GB)	{6400; 12800; 25600}	{100; 200; 400; 800; 1600}
	Memory (GB)	{256; 512; 1024}	{2; 4; 8; 16; 32; 64}
Links	Bandwidth (Mbps)	{800; 1200}	{34.368 139.264}

Table 3.1 – Physical and virtual network resource parameters

3.2. Algorithm description

The following mapping algorithm is an enhancement of the mapping algorithm presented by [62], and has the objective to improve the embedding of VIs (with cloud elements, nodes and

servers). It is important to notice that the base mapping algorithm has the objective of balancing the load in the physical infrastructure; the maximization of the embedding of VIs is in a second plan (as a consequence of the load balancing). We imposed a condition that ensures that each different candidate can only host one VM from the same VI request, in order to avoid the over-allocation of the physical infrastructure elements. The pseudo-code of the mapping algorithm is in the end of this section, and the modifications implemented by this work are highlighted in grey:

- **Candidate list creation (lines 1-23)** – The first part of the algorithm is the creation of the candidate list for the virtual nodes (routers and servers). The candidate's list creation can be divided in three different stages. First, it compares the requested hardware features HDD memory, RAM, CPU frequency, with the hardware features of each node from the physical infrastructure, limiting the comparison with the correspondent type of the nodes (virtual server with server and virtual routers with routers). The physical nodes that support the demanded characteristics are added to the virtual node list of candidates (lines 1-3). Once the VI is a set of connected virtual nodes, the second stage evaluates the capacity of the previous candidates to connect with the neighbors' virtual nodes¹ candidates, proposed as interdependence mapping by [62] (lines 4-9). This stage analyzes also the QoS requirements in the VI that the candidates have to respect, as delay and latency. Finally, in the third stage it is performed a first level try of the candidates to strengthen the list of candidates in order to achieve better mapping performance (lines 10-23).
- **Substrate stress calculation (lines 24-38)** – The second part is the calculation of the physical nodes and links stress (lines 24-38). This value is used in the third part, where the selection of candidates is performed. The objective of this stress is to balance the occupation levels in the substrate.
- **Decision process (lines 39-58)** – The third part is the decision process that makes the allocations of the VMs to the physical nodes. The first stage of this process is a safety routine that assures the condition of hosting (lines 40-45). The selection of the candidates for every virtual node is made by choosing the candidate that presents the higher potential from each candidates list. The candidate potential is inversely proportional to the multiplication of the candidate stress with the average of the path costs necessary to connect the candidate in analysis, with all candidates of the neighbor's virtual nodes (line 46-50). The node stress is calculated before this process and the path cost is calculated, using a modified version of the shortest path finding algorithm *Dijkstra* (line 48), *HOP_Dijkstra* that is explained in the section 3.4.2. After finding the candidate with higher potential, the algorithm removes that candidate from the other virtual nodes candidates lists, and removes the rest of the candidates from the virtual node in hosting process, before passing to the selection of the candidate of the next virtual node. The algorithm has a safety mechanism that cancels the last decision if, because of that, the entire list of

¹ It is considered neighbors virtual nodes, the virtual nodes from the same VI that have a connection in common.

candidates became inadequate, the “backtracking mechanism” proposed by [62] (lines 51-57).

- **Virtual link mapping (lines 59-65)** – Finally, the fourth part is the link allocation. In this part the mapping algorithm uses the same path finding algorithm as before, *HOP_Dijkstra*, in the calculation of the candidate potential, in order to map every virtual link that is presented in the VI request. It is needed to use the same path finding algorithm to be coherent with the candidate selection.

Algorithm 1: Pseudo-Code of the Enhanced Heuristic Algorithm

```

input : Substrate (Substrate Network) ,  $V_{Request}$  (Requested VI)
output:  $V_{Map}$  (Mapped VI)

1  foreach Node  $n$  in  $V_{Request}.Nodes$  do
2     $n.Candidates(i) = FindCandidates(Substrate.Nodes)$  ;
3  end
4  foreach Link  $v$  in  $V_{Request}$  do
5     $PossiblePath(v) = FindPossiblePath()$ ;
6  end
7  foreach Node  $i$  in  $V_{Request}$  do
8     $RemoveCandidatesWithoutAnyPossiblePathToOneVirtualNeighbor(i)$ ;
9  end
10  $SaveData(ListsOfCandidates, PossiblePath)$ ;
11 foreach Node  $n$  in  $V_{Request}.Nodes$  do
12   foreach Candidate  $j$  in  $n.Candidates$  do
13      $RemoveNonSelectedCandidates(n)$ ;
14      $RemoveSelectedCandidateFromOtherNodes(j)$ ;
15     if  $NumberOf(x.Candidates) == 0, \forall x \text{ in } V_{Request}.Nodes$  then
16        $RestoreData(ListsOfCandidates, PossiblePath)$ ;
17        $RemoveCandidate(j, n.Candidates)$ ;
18        $SaveData(ListsOfCandidates, PossiblePath)$ ;
19     else
20        $RestoreData(ListsOfCandidates, PossiblePath)$ ;
21     end
22   end
23 end
24 foreach Link  $i$  in Substrate.Links do
25    $S_{LS}(i) = CalcLinkStress(Link(i))$  ;
26 end
27 foreach Node  $i$  in Substrate.Nodes do
28   if  $Node(i).Server == true$  then
29      $S_{Ni} = CalcServerStress(Node(i))$  ;
30   else
31      $S_{Ni} = CalcRouterStress(Node(i))$  ;
32   end
33 end
34 foreach Node  $i$  in Substrate.Nodes do
35   if  $S_{Ni} == 0$  then
36      $S_{Ni} = MinStressValue(S_N \neq 0)$  ;
37   end
38 end
39 while  $\exists Node x \text{ in } V_{Request}.Nodes \mid NumberOf(Node(x).Candidates) > 1$  do
40   foreach  $NumberOf(n.Candidates) == 1, \forall n \text{ in } V_{Request}.Nodes$ 

```

```

41     if n.Map == 0 then
42         RemoveSelectedCandidateFromOtherNodes(n.Candidate);
43         n.Map = n.Candidate
44     end
45 end
46 n = SelectUnmappedNodeWithLessCandidates( $V_{Request}.Nodes$ );
47 foreach SourceCandidate v in n.Candidates do
48      $\pi(v) = \text{CalculateNodePotential}(v)$ ;
49 end
50 n.Map = v :  $\pi(v) = \min(\pi)$ ;
51 SaveData(ListsOfCandidates, PossiblePath);
52 RemoveNonSelectedCandidates(n);
53 RemoveSelectedCandidateFromOtherNodes(v);
54 if NumberOf(n.Candidates)==0,  $\forall n$  in  $V_{Request}.Nodes$  then
55     RestoreData(ListsOfCandidates, PossiblePath);
56     RemoveCandidate(n,n.Map);
57 end
58 end
59 foreach Node n in  $V_{Request}.Nodes$  do
60      $V_{Map}.Nodes \cup n$ ;
61     foreach Link k connected to n do
62         ConnVNode=GetLinkDestination(k);
63          $V_{Map}.Links = \text{HOP\_Dijkstra}(n.Map, ConnVNode.Map)$ ;
64     end
65 End

```

3.3. Candidates List

A candidate starts to be the node of the physical infrastructure that has the adequate hardware features to host the virtual node (higher or equal characteristics in the number of cores, CPU frequency, memory and storage, if it is a server). Since the purpose is to map VIs and not independent virtual nodes (VMs), the hardware feature is not enough to make the physical nodes good candidates. Furthermore, the candidate needs to have a possibility of link (path) that supports the virtual link bandwidth and QoS requirements, with at least one candidate of all the neighbors' virtual nodes to be accepted as a legal candidate. This last condition makes a remission of a candidate a more complex process than a simple direct list removal. A unique removal of the candidate can cause the entire candidates list to be inadequate.

With these requirements, the candidates list is narrowed down, but it still remains fragile. We propose a better oversight of the candidate list. The optimal solution would be to extend that interdependency mapping to all the nodes of the virtual node. We therefore propose to use the proprieties of the selection, since a selection of a candidate to host a virtual node is pursued by the removal of the rest of the candidates for that node, and by the removal of the selected candidates from other virtual nodes lists. These removals affect the rest of the candidates because of the interdependence condition. Basically, we perform a first level try selection, through every candidate, to eliminate all the pseudo-candidates that provoke the immediate failure of the selection. To strengthen the selection, we introduce a check routine in the beginning

of the decision process cycle. With this check routine, we assure that it will not be hosted more than one virtual node from the same VI in each physical infrastructure nodes.

3.3.1. Candidates first selection try

In this part the candidates are tested individually. Before the test process initiates, it is saved the list of candidates and the information of the current possible connections. Afterwards, the cycle performs the selection of every candidate and makes the necessary subsequent removals. When the list of candidates becomes inadequate because of the previous process, this cycle updates the list of candidates by removing that pseudo-candidate. In fact this test goes further than the interdependence mapping, because it takes in consideration every virtual nodes of the request instead of just the neighbors' virtual nodes. The disadvantages of this process are the time consumption and the lack of response in the cases the selected candidates are not shared from different virtual nodes lists. The first disadvantage is inconvenient, but it does not compromise the characteristics of the heuristic approach. The last disadvantage cripples the mechanism by not letting it reach every virtual node of the VI in those specific cases.

3.3.2. Check Routine

Due to the fact that each virtual node has to be hosted in different nodes of the physical infrastructure, this check routine becomes important. To better understand its importance, different scenarios will be presented below in which the base algorithm [62] leads to a failure of selection, and with the use of the check routine the problem is overcome. Note that each virtual node can end with only one candidate due to three different reasons:

- As a consequence of the decision process.
- As a consequence of the decision process related with another virtual node.
- As a consequence of the candidates requirements (hardware features and connections for the candidates of the neighbors' virtual nodes).

The first consequence is not problematic because the decision process certifies that the candidate chosen will not be available for other virtual nodes before proceeding to the next selection. The other two consequences can provoke the premature failure of the selection.

In the following tables with the examples, we will consider the virtual nodes, VM1 VM2 VM3 VM4, belonging to the same VI request in mapping process, and we will not distinguish between routers and servers. The candidates list presented, composed by the physical nodes A, B, C, D, will be assumed as the final result of the candidate list creation, iteration #1 in all the tables.

		Virtual Nodes			
		VM1	VM2	VM3	VM4
Iteration moments	#1	A, B	A, C, D	A, B, D	A, C
	#2	A	C, D	B, D	C
	#3	A	C	D	C

Table 3.2 – Base algorithm decisions - example 1

For the first example, in Table 3.2, the base algorithm gives priority to the virtual nodes that have lower number of candidates, larger than one (VM1 and VM4). We will assume that the virtual node chosen for the selection is the VM1, and the result of that selection is the candidate A. With that allocation, the candidate A is removed from the other virtual nodes candidate's lists (VM4), and the rest of the candidates are removed from the VM1, in this case it is just the candidate B. The state of the selection is represented in iteration #2. In this moment, the VM2 and VM3 are in the position of being chosen to perform the selection. Assume the VM3 is chosen and the result of its selection is the candidate D. After the removals, the decision process is finished because every virtual node has only one candidate, iteration (3). It is noticeable that, when the VM4 has only the candidate C, the algorithm did not perform any action, which resulted in the sharing of the same candidate by the VM2 and VM4, highlighted in the table, and provoking a failure of mapping.

		Virtual Nodes			
		VM1	VM2	VM3	VM4
Iteration moments	#1	A, B	A, C, D	A, B, D	A, C
	#2a	A	C, D	B, D	C
	#2b	A	D	B, D	C
	#2c	A	D	B	C

Table 3.3 – Algorithm with check routine decisions - example 2

The example 2, in Table 3.3 (same conditions of the previous example), assumes that the order of selection of the virtual nodes is the same used in the previous example, in Table 3.2. With the check routine implemented, the VM4 is not ignored and assures that the candidate C is exclusive (2a). As a result of that, the VM2 has only the candidate D and performs the same process (2b). In this case, the success of the selection is assured and performed with a lower number of iterations, since the selection is assured inside the check routine in the #2 iteration moment.

		Virtual Nodes		
		VM1	VM2	VM3
Iteration moments	#1	A, B	A	B, C
	#2	A	A	B

Table 3.4 – Base algorithm decisions - example 3

The example 3, in Table 3.4, is similar to the second part of the previous examples. The base algorithm will ignore the VM2 because it has only one candidate. Assuming the VM3 is chosen and the candidate B is selected, iteration moment #2, the final result will be the same as the one

in Table 3.2. Since the sharing of a candidate is not allowed, the mapping of the current VI would fail.

		Virtual Nodes		
		VM1	VM2	VM3
Iteration moments	#1a	A, B	A	B, C
	#1b	B	A	B, C
	#1c	B	A	C

Table 3.5 – Algorithm with check routine decisions - example 4

The Table 3.5 represents the stages of the check routine for the same conditions of the example 3, in Table 3.4. In this case, the decision process only needs one iteration moment, since each virtual node has only one candidate before exiting the check routine, during the iteration moment #1. Here, the selection process has finished successfully and with faster performance again.

3.3.3. Performance Evaluation

In this sub-section we study the performance of the base heuristic (Base-H), enhanced heuristic with the first try routine (Enhanced-H-FT), check routine (Enhanced-H-CR) implemented individually, and the enhanced heuristic (Enhanced-H) with both modifications together. We study the performance of the algorithms ranging the size of the physical infrastructure from 15 to 55 nodes (15, 25, 35, 45 and 55). The analysis comprises 10 runs (10 different substrates) with 500 time units with a VI arrival rate of 2 per time unit (1000 VI arrivals per run with a total of 10000 VI arrivals). The study focuses on the analysis of the acceptance ratio, revenue ratio, occupation of bandwidth of the substrate and node/link usages. All values in the graphics present a mean of the runs with a Confidence Interval (CI) of 95%.

The aspects analyzed are the performance in terms of VI's acceptance and its correspondent revenue, in Figure 3.1. The performance of each individual improvement and with both improvements together is very similar; however, it is noticeable a gap from the base heuristic for all the physical infrastructure nodes, especially noticeable for the substrate of 25 nodes. The proposed enhancements present better values of acceptance and revenue comparing with the base algorithm. Besides de substrate of 35 nodes, the Enhanced-H presents the best result. The revenue ratio accompanies the tendency of the acceptance ratio. The algorithm that presents better acceptance ratio also presents better revenue, which shows that the average quality (characteristics) of the accepted VI is approximately the same for all the approaches.

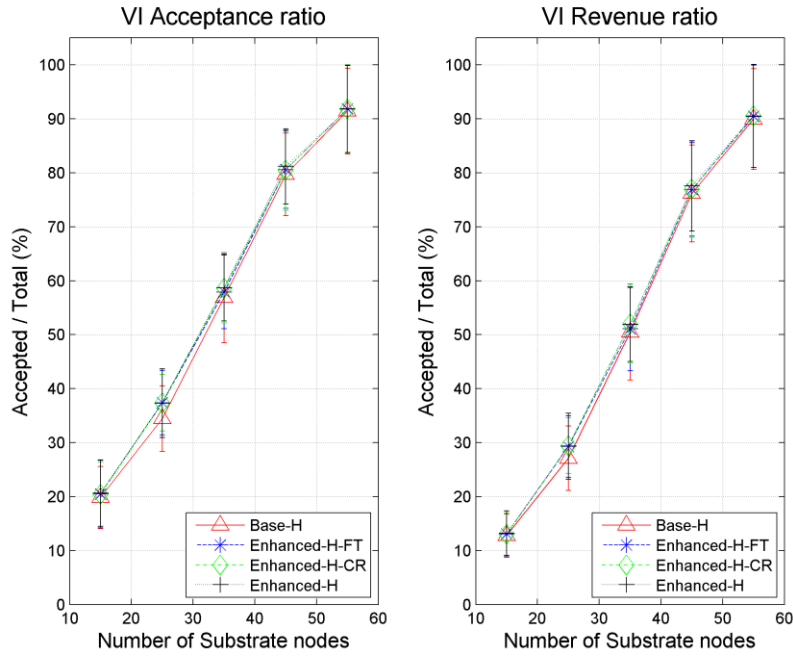


Figure 3.1 – VI acceptance and revenue ratio - candidates list

Figure 3.2 shows the bandwidth occupation of the physical infrastructure. Here it is noticeable again the gap between the base heuristic and the rest. The Base-H presents lower occupation levels, because the level of occupation is directly related with the acceptance ratio, and the Base-H presents also lower acceptance ratio. Since all the algorithms in this stage use the same *Dijkstra* approach to find the paths in order to map the links, higher acceptance means higher occupation of the physical infrastructure. In particular, the Enhanced-H is the approach that presents the lower occupation levels between the proposed algorithms, except for the substrate of 25 nodes.

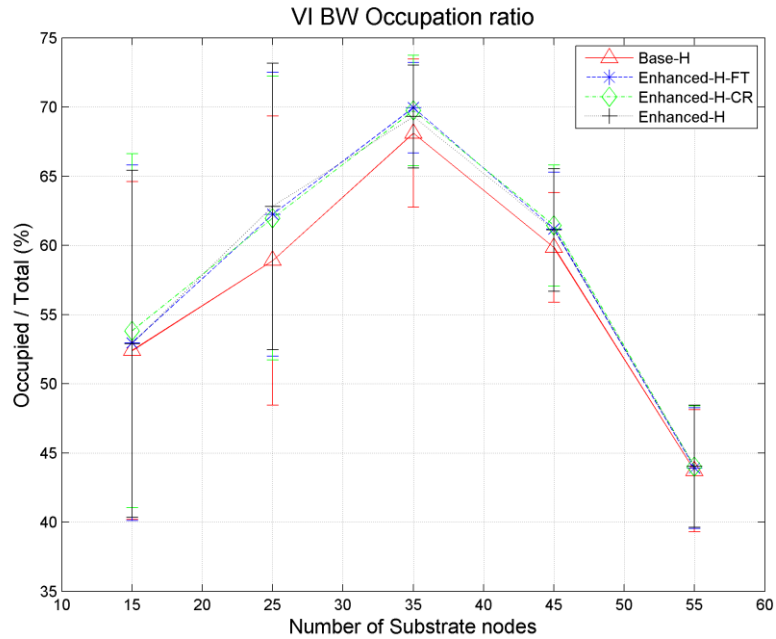


Figure 3.2 – Physical Infrastructure link occupation ratio - candidates list

Figure 3.3 depicts the nodes and links consumption. Besides the substrate with 15 nodes, where the difference is small, all the consumption for the different approaches is similar. This shows that the usage does not depend on the quality of the candidates list, and the consumption is just related with the path finding algorithm used. For the bigger substrate nodes, the power consumption for all the approaches is near 100 percent for both, links and nodes.

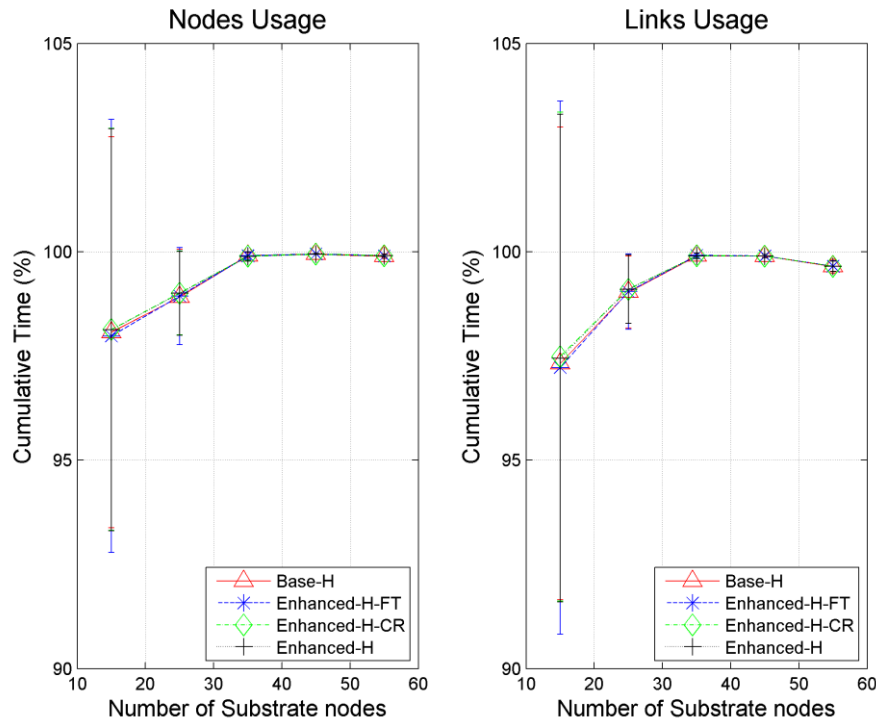


Figure 3.3 – Physical Infrastructure usage - candidates list

Although we conclude that the performance is similar between the enhanced heuristics, for the upcoming approaches we will use the mapping approach that implements both the modifications Enhanced-H (first try routine and check routine), because it is the approach that most benefits brings to the operator, and also assure that we get the best candidate list possible. These results suffer from the randomness of being an online scenario, and with this decision, we assure that we use the most reliability algorithm.

3.4. Decision Process

The previous chapter has shown the importance to get the best candidate list possible to improve the algorithm performance. Despite the importance of enhancing the candidates lists, the decision process is the most important part of the mapping algorithm, because it is the place where the virtual nodes are allocated. This part of the algorithm does the selection of the candidates for all the virtual nodes of the VIs. It is important to realize that the decision process is a sequential process, one virtual node at a time, so every decision influences the upcoming

selections because of the condition of hosting², and since the next virtual node neighbors will only consider the selected candidates of the previous iterations for the calculation of the candidate potential.

In order to maximize the embedding of VIs requests, it is needed to improve the selections made by the decision process. We propose modifications to both elements used to calculate the candidates' potential. The path cost is the most important of the two elements, because it is related directly with the bandwidth of the physical links and because it presents a sharper variation. We suggest to use the sharper variation and to amplify it to become a better metric of comparison between candidates. With respect to the node stress, we assure that the element is never null.

3.4.1. Node Stress

Although the link bandwidth of the physical infrastructure is the main constraint of the embedding of VIs, the node stress is needed in the decision process to increase the occupation of the physical nodes in a way to maximize the embedding. In the base algorithm this factor could withdraw the selection by neutralizing the potential calculation in the decision process, when the physical nodes are not hosting any virtual nodes. Those moments are achieved by the departure of virtual networks or by periods the network is not overlooked. In those cases, the potential of those candidates would be infinite independently of the path cost that they would have for the neighbors' virtual nodes.

We assure that the node stress of the candidates is never zero in order to perform better decisions. More than that, we suggest that, after one candidate presents a value of stress different than zero, all the physical nodes that do not host any VM will be virtually given the minimum value of node stress between the nodes that are hosting virtual nodes. With this measure, the physical nodes that were never chosen to allocate virtual nodes will be in the first place to do it, because they will have the lower level of stress. This measure will always take in consideration the path cost in their potential calculation.

3.4.2. HOP_Dijkstra

The *Dijkstra* algorithm is used to find the shortest physical path for a virtual link between two known nodes from the same substrate, source node and destination node. The result of this search is used to quantify the candidates in the decision process together with the node stress. In the base algorithm, the *Dijkstra* would give the cumulative cost of the links stress that forms the discovered shortest path. It is evident that this method does not take in consideration the number of links used, but simply discover the path less stressed. This fact constraints the success of embedding VIs in the physical infrastructure, because those paths result in an unnecessary occupation of the physical infrastructure (higher number of links). We propose to take in consideration the amount of links used when the algorithm is finding the path. The approach we

² Every virtual node has to be hosted in a different physical node (candidate)

suggest imposes weights to the hops in addition to the physical infrastructure occupation by a factor K of the virtual link in concern. Equation 1 and 2 reflects the cost calculation of mapping a virtual link, for the previous *Dijkstra* approach and for our *Dijkstra* approach, respectively. $CFSP_COST$ and EXP_COST represents the cost, H represents the number of hops (number of substrate nodes besides the source node and destination node), $L(i,j)$ represents the link stress between the node i and the node j , P represents the sorted list of nodes that the link traverses, K represents a constant value, and VL represents the bandwidth capacity of the virtual link.

$$CFSP_COST = \sum_{n=1}^H L_{(P_n, P_{n+1})} \quad (1)$$

$$HOP_COST = \sum_{n=1}^H L_{(P_n, P_{n+1})} + \sum_{n=1}^H K.(n-1).VL + VL \quad (2)$$

In order to understand the difference of behavior between both *Dijkstra* approaches we will present some examples. The first example shows the range of values that each approach works, and the second example shows a practical case of making a selection. For all the examples, assume that all the candidates have the same node stress higher than zero.

For the first example, we assume the VM1 has only the candidate A, the VM2 has only the candidate D, and VM1 has only one connection with the VM2 in the VI request.

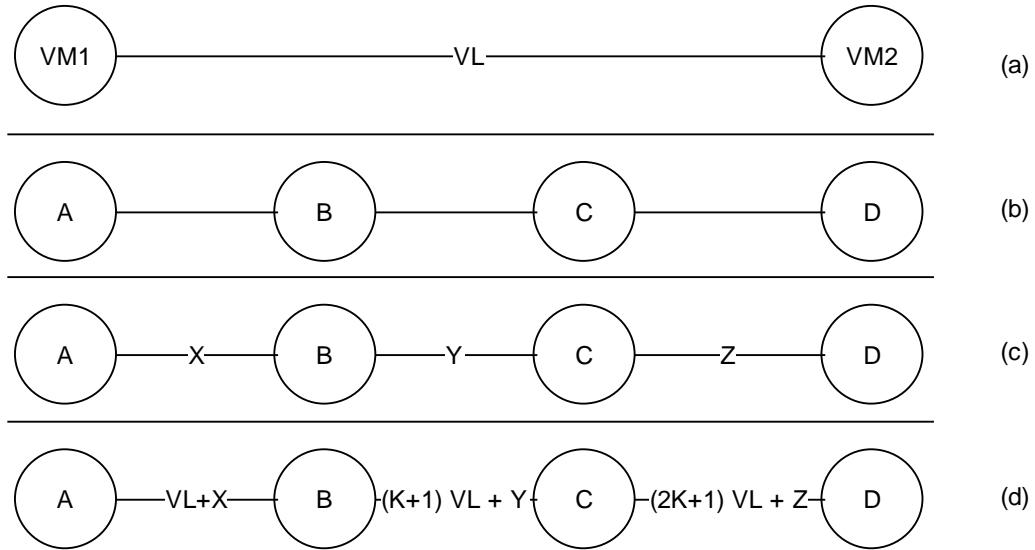


Figure 3.4 – Virtual link path finding

The virtual link is shown in the Figure 3.4 (a). For this example, we assume the *Dijkstra* solution is the same for both approaches in which the path is concerned, Figure 3.4 (b). The $CFSP_COST$, Figure 3.4 (c), would be the sum of the link stress weighs presented in that moment in the three connections (A—B + B—C + C—D); in this case the final path cost is $X+Y+Z$. For the HOP_COST , Figure 3.4 (d), the final path cost is $(3K+3)VL+X+Y+Z$. It is noticeable that the range of the path cost increases exponentially with the increase of hops. Since the candidate potential is inversely proportional with the multiplication of the path cost with the node stress, the candidates that

provide fewer hops for the link will be selected. The link stress works in this formula as a tiebreaker in the cases that candidates have the same average of hops.

The second example represents a selection decision for a VM1 between its two candidates, A and G. The VM1 has only one connection with the VM2, with a capacity of 34 Mbytes, see Figure 3.5 (e). The VM2 has only one candidate F. The decision process aims to decide which candidate, A or G, is more appropriated to host the VM1. Assume the constant K has the value of 100.

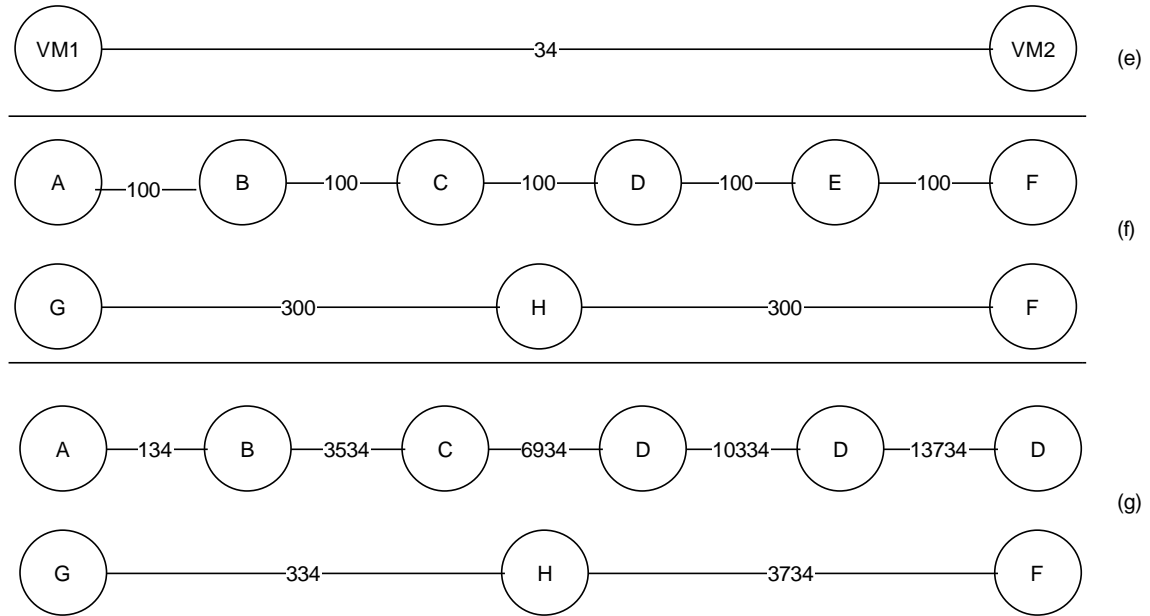


Figure 3.5 – Virtual link path finding for the both candidates A and G

For the *Dijkstra* approach used in the base mapping algorithm, the candidate A would be a better candidate than G because it would have a path cost of 500 (100+100+100+100+100) instead of 600 (300+300). Since the potential of a candidate is inversely proportional with the path cost multiply with the node stress, the candidate that presents lower path cost has higher potential and is selected, because every physical node has the same node stress. In this case it would be selected the candidate A, and that decision would provoke a larger occupation of the physical infrastructure by 102 Mbytes (more three links are reserved), in comparison to the case of selection of candidate G. For the *Dijkstra* approach proposed in this work, the candidate G would be undoubtedly chosen, because it would have a path cost of 4068 against the path cost of 34670 of the candidate A.

In summary, we make the efficient use of the links an even more important aspect than the base algorithm for the candidate selection. Our approach does not allow choosing candidates that will consume a much higher number of links, even if the physical infrastructure in that zone has a lower stress value. Besides, in the calculation of the candidates potential, this approach is also used in the final link mapping to maintain the coherence.

3.4.3. Performance Evaluation

In this sub-section we study the performance of the previous enhanced heuristic (Enhanced-P-H), the previous enhanced heuristic with the node stress enhancement (Enhanced-P-H-NS), the enhanced heuristic with the *HOP_Dijkstra* approach with the node stress (Enhanced-H), and without the node stress (Enhanced-H-HD). We range the size of the physical infrastructure from 15 to 55 nodes (15, 25, 35, 45 and 55). The analysis comprises 10 runs (10 different substrates) with 500 time units with a VI arrival rate of 2 per time unit (1000 VI arrivals per run with a total of 10000 VI arrivals). The study focuses on the analysis of the acceptance ratio, revenue ratio, occupation of bandwidth of the substrate and node/link usages. All values in the graphics present a mean of the 10 runs with a CI of 95%.

The aspects analyzed are the performance in terms of VI's acceptance and its correspondent revenue, in Figure 3.6. The Enhanced-H has a better performance from the rest for all the substrates analyzed, in terms of acceptance and revenue. It is interesting to see that the node stress just became relevant together with the new *Dijkstra* approach. This makes sense since the previous mapping objective was to balance the load in the substrate, and towards that objective nothing is better than imposing the physical nodes that do not host any virtual node with infinite potential to force their selection. The revenue ratio reflects directly the acceptance ratio. The measures that present better acceptance also present better revenue.

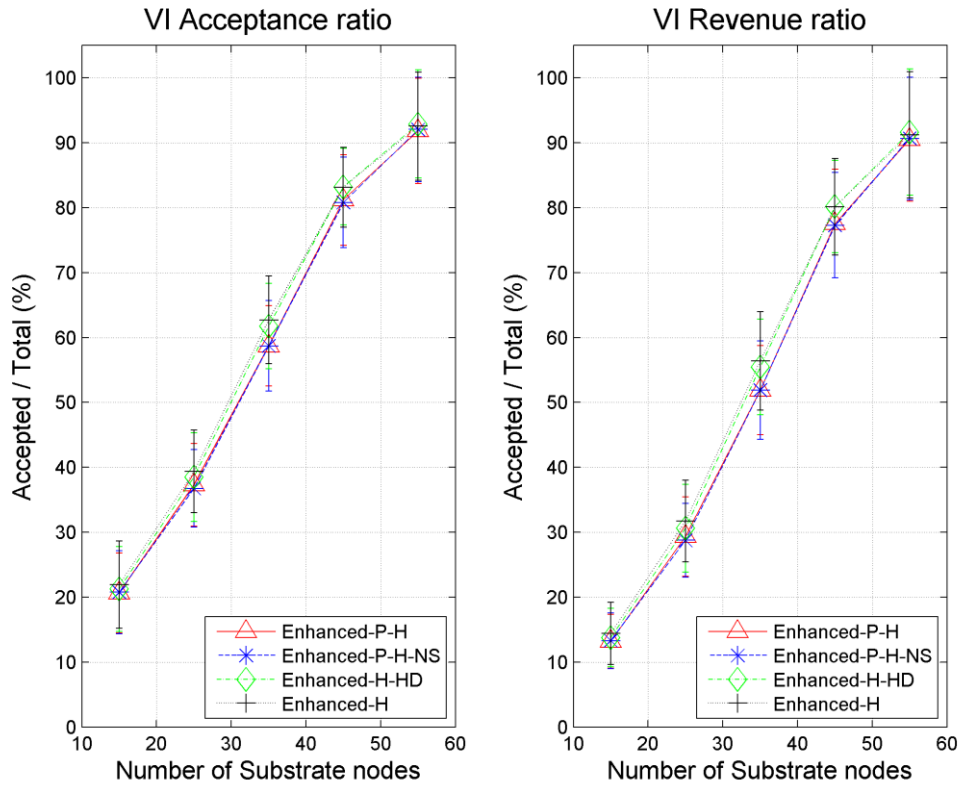


Figure 3.6 – VI acceptance and revenue ratio - decision process

In Figure 3.7 we observe the bandwidth occupation of the physical infrastructure. Here the previous conclusion remains equal. It is the feature with the larger gap between the approaches that uses different path finding algorithms. The Enhanced-H and Enhanced-H-HD algorithms present lower occupation levels between 3% to 5% for the smaller substrates, and 8% to 10% for the bigger substrates from the Enhanced-P algorithms. This fact is justified because the Enhanced-H algorithms use a different path finding approach. The algorithms that use the *HOP_Dijkstra* approach show better substrate occupation, and at the same time, better VIs acceptance ratio and revenue ratio.

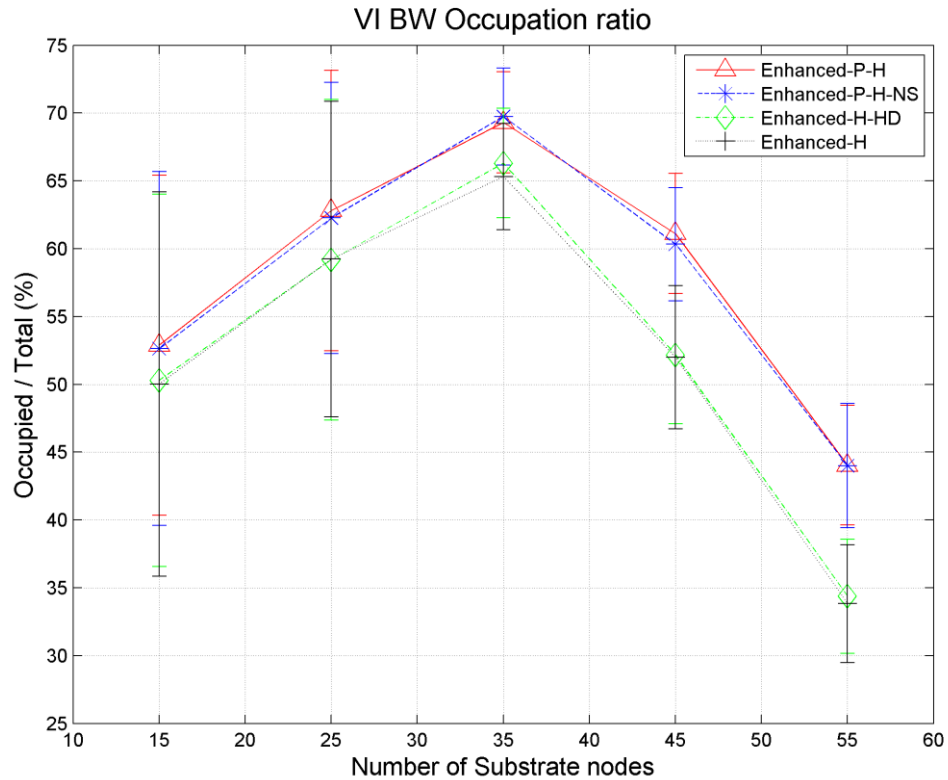


Figure 3.7 – Physical Infrastructure link occupation ratio - decision process

Figure 3.8 shows the nodes and links consumption. The node stress role is clearer here. With the node stress implemented, the Enhanced-H algorithm presents lower power consumption while the remaining features are improved. The tendency of the link usage indicates that, with larger physical infrastructures, the gap between the algorithms that use the *HOP_Dijkstra* approach and the rest will increase.

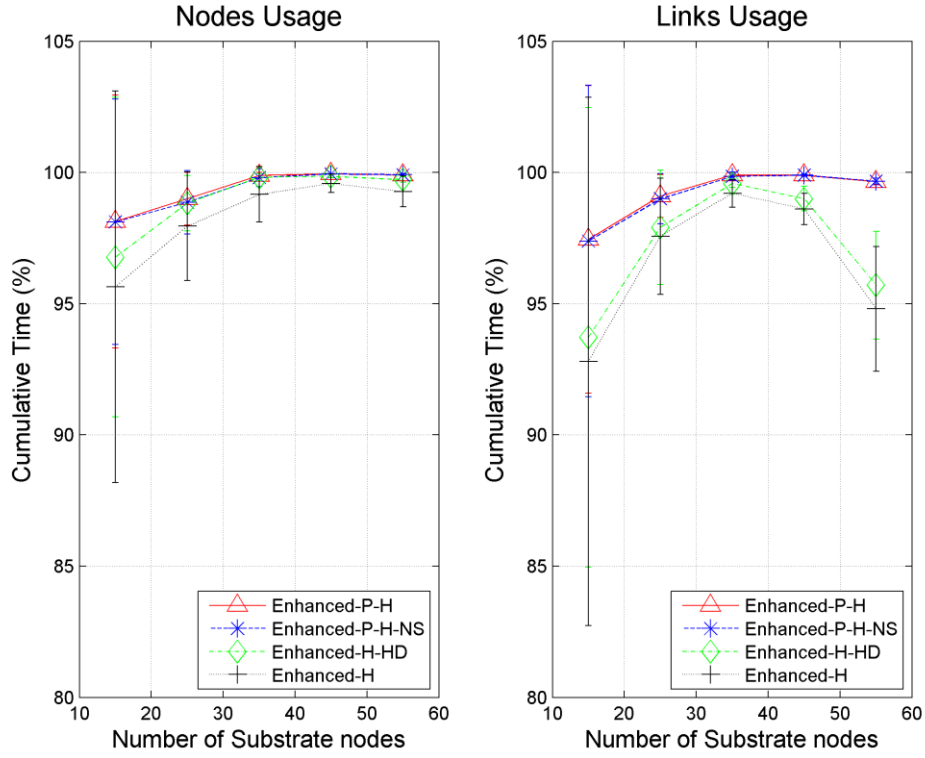


Figure 3.8 – Physical Infrastructure usage - decision process

We conclude that the *HOP_Dijkstra* approach is better to maximize the profits of the operator. In all the aspects it shows improvements: better acceptance and revenue ratios and lower substrate resource occupation and energy consumption. For the upcoming simulations, the enhanced heuristic is composed by the base-algorithm and all the modifications mentioned in the past two sub-chapters, 3.3 and 3.4 (Enhanced-H).

3.5. Overall Performance Evaluation

In this sub-section we study the performance of the base algorithm and the last version of the enhanced heuristic. The analysis comprises 10 runs (10 different substrates) with 500 time units with a VI arrival rate of 2 per time unit (1000 VI arrivals per run with a total of 10000 VI arrivals). We range the size of the physical infrastructure from 15 to 55 nodes (15, 25, 35, 45 and 55). Moreover we fix the substrate of 35 nodes and change the arrival rate between 1 and 5 (1 2 3 4 and 5). The study focuses on the analysis of the acceptance ratio, revenue ratio, occupation of bandwidth of the substrate and node/link usages. It is also studied the behavior of both algorithms over the time for the infrastructure usage. All values in the graphics present a mean of the 10 runs with a CI of 95%.

The performance of the heuristic algorithm with the *HOP_Dijkstra* approach is better in every aspect analyzed: acceptance and revenue ratio for the different substrates and VI arrivals rate, as observed in Figure 3.9 and Figure 3.10. However, for large substrates the gap between the base heuristic algorithm and the enhanced heuristic algorithm gets closer. For the difference of VI

arrivals rate, the gap between algorithms stays constant. When the size of the physical infrastructure is large, the Base-H algorithm can compete with the Enhanced-H algorithm in terms of acceptance and revenue.

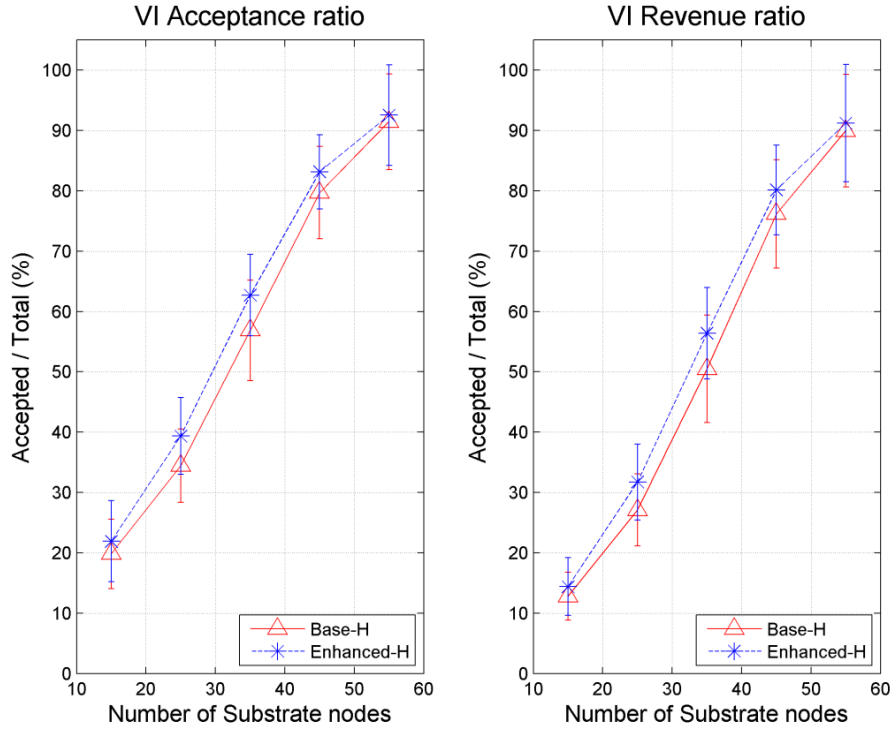


Figure 3.9 – VI acceptance and revenue ratio

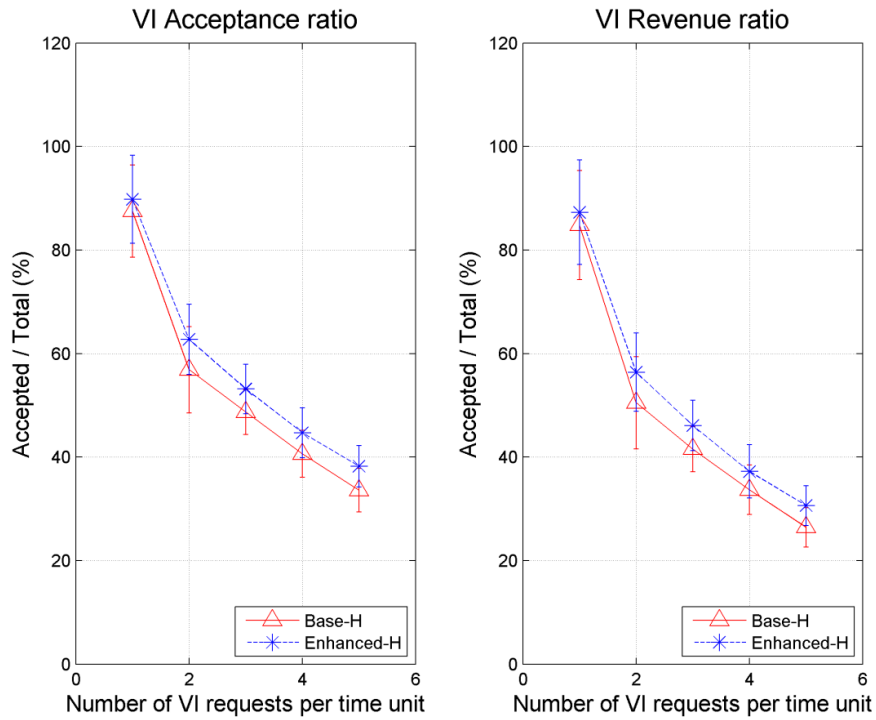


Figure 3.10 – VI acceptance and revenue ratio - request rate variation

Figure 3.11 shows the level of occupation of the physical infrastructure for the different substrates and VIs arrivals rates. Besides the substrate of 25 nodes and the arrival rate of 5 for the substrate of 35 nodes, the enhanced heuristic algorithm presents always lower occupation. Besides the enhanced heuristic algorithm presenting better performance in the embedding of VIs, it also reduces the occupations of the physical infrastructure. This is related with the path finding approach utilized to chose the candidates and map the virtual links, proving that the *Hop-Dijkstra* is more appropriated than the normal *Dijkstra* to maximize the profits of the operator.

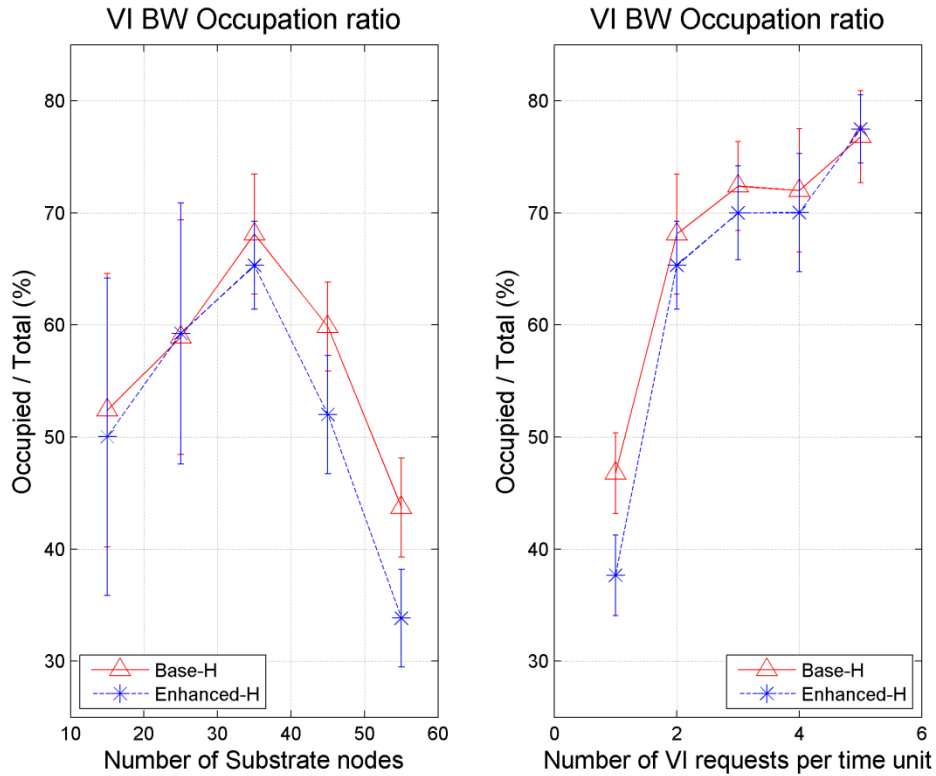


Figure 3.11 – Physical Infrastructure link occupation ratio

Figure 3.12 and Figure 3.13 show the usage of the infrastructure elements nodes and links. For both cases, the enhanced heuristic shows lower utilizations, especially noticed for the link usage in the substrate of 15 nodes and 55 nodes, where it presents approximately less 5% of use comparatively with the base algorithm. It is noticeable with the increasing of the rates the utilizations get close between both algorithms because, for the same period of time, there are more VIs hosted and the enhanced heuristic algorithm gets completely stressed, as the base heuristic algorithm always is.

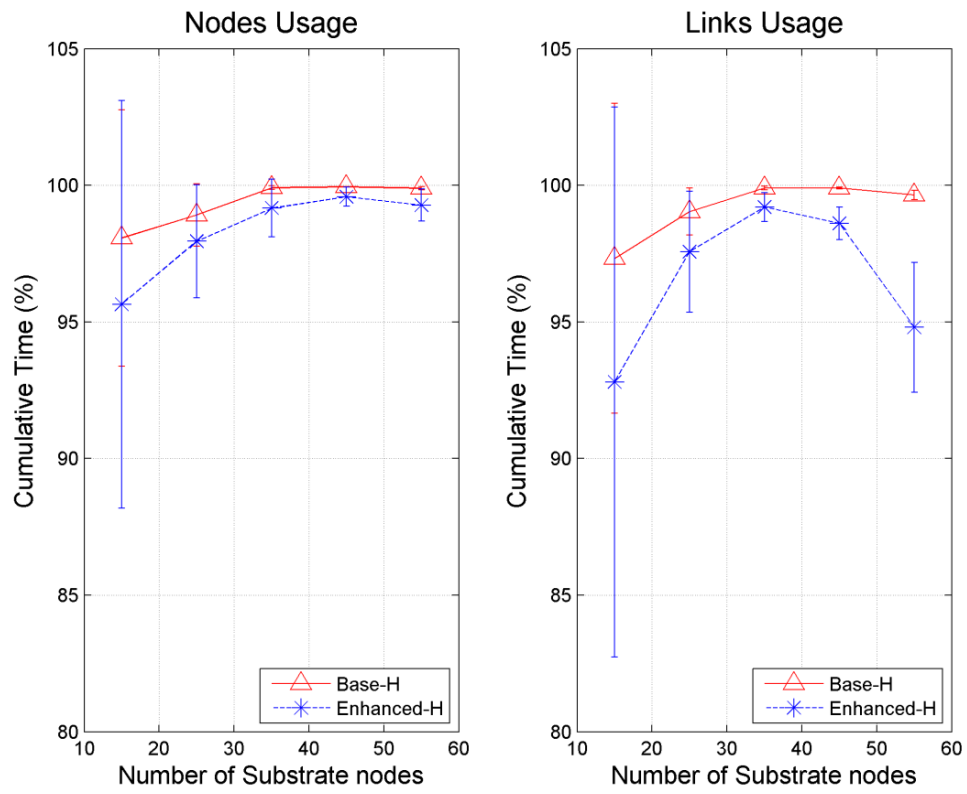


Figure 3.12 – Physical Infrastructure usage

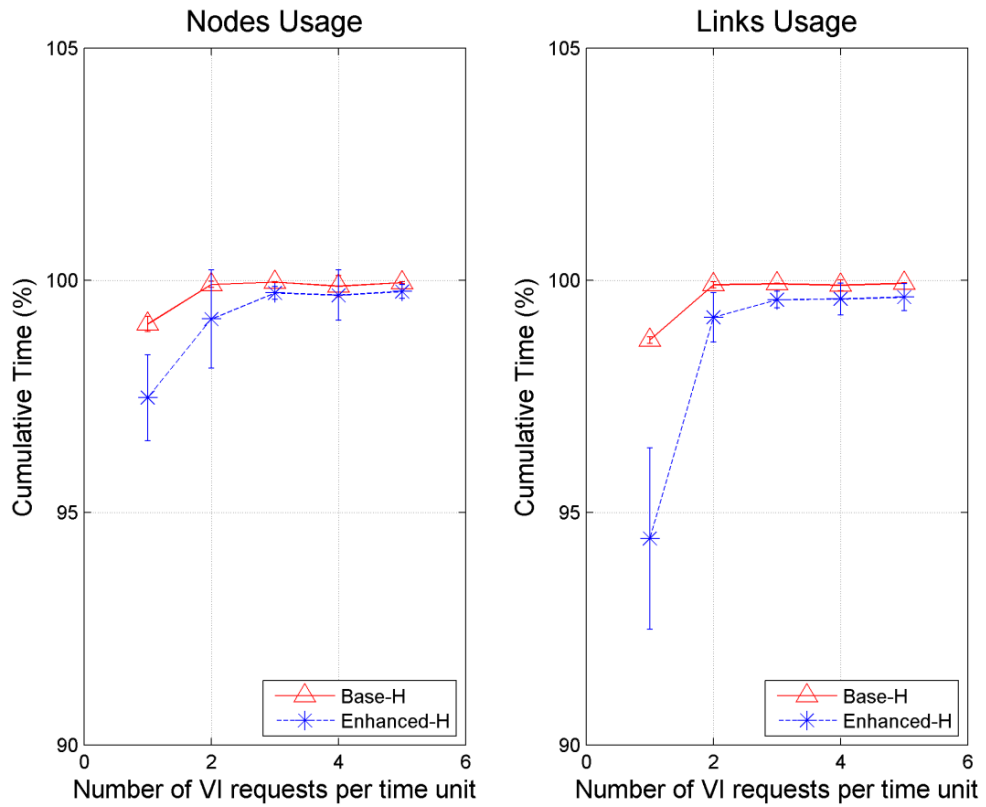


Figure 3.13 – Physical Infrastructure usage - request rate variation

The performance of the infrastructure consumption over time is shown in the Figure 3.14, Figure 3.15, Figure 3.16, Figure 3.17 and Figure 3.18. In all these figures, it is noticeable that the enhanced algorithm has the same tendencies of the base algorithm. The zones where the base heuristic algorithm shows an inconstant behavior are the same of the enhanced heuristic algorithm. We can then conclude that all the differences of performance mentioned above have a uniform appearance, and do not result by a drastic change of behavior of the algorithm. The time units of the different simulations are not the same, due the fact that the VI arrivals and departures are produced based on Markov-modulated inter-arrival and inter-departure times.

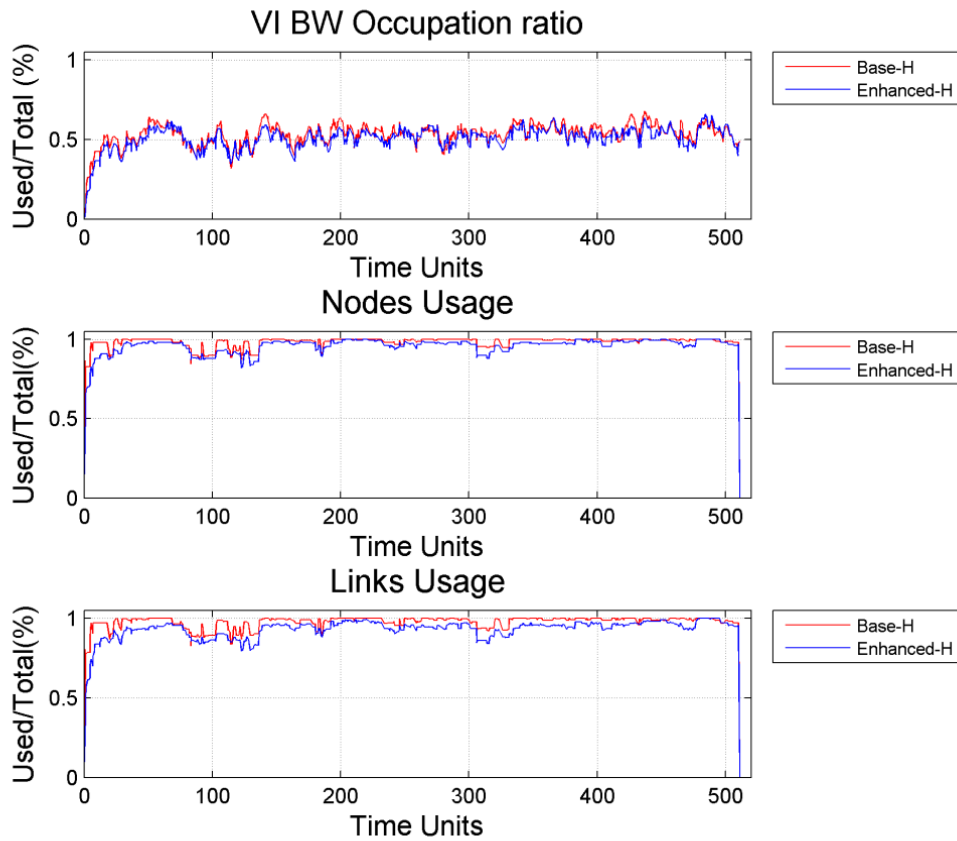


Figure 3.14 – Physical Infrastructure occupation (Substrate 15 nodes)

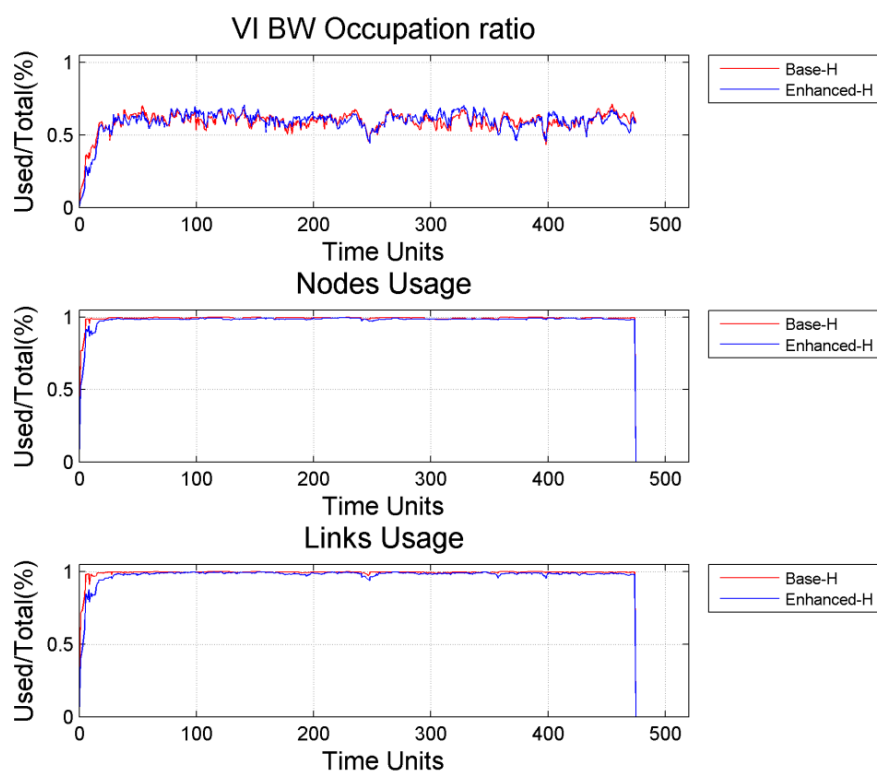


Figure 3.15 – Physical Infrastructure occupation (Substrate 25 nodes)

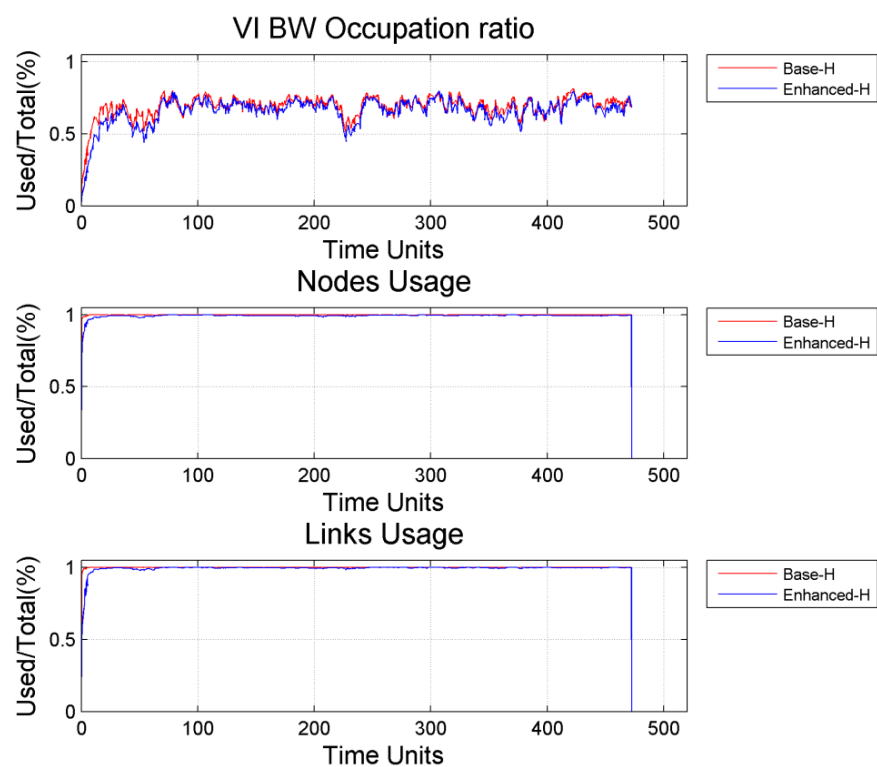


Figure 3.16 – Physical Infrastructure occupation (Substrate 35 nodes)

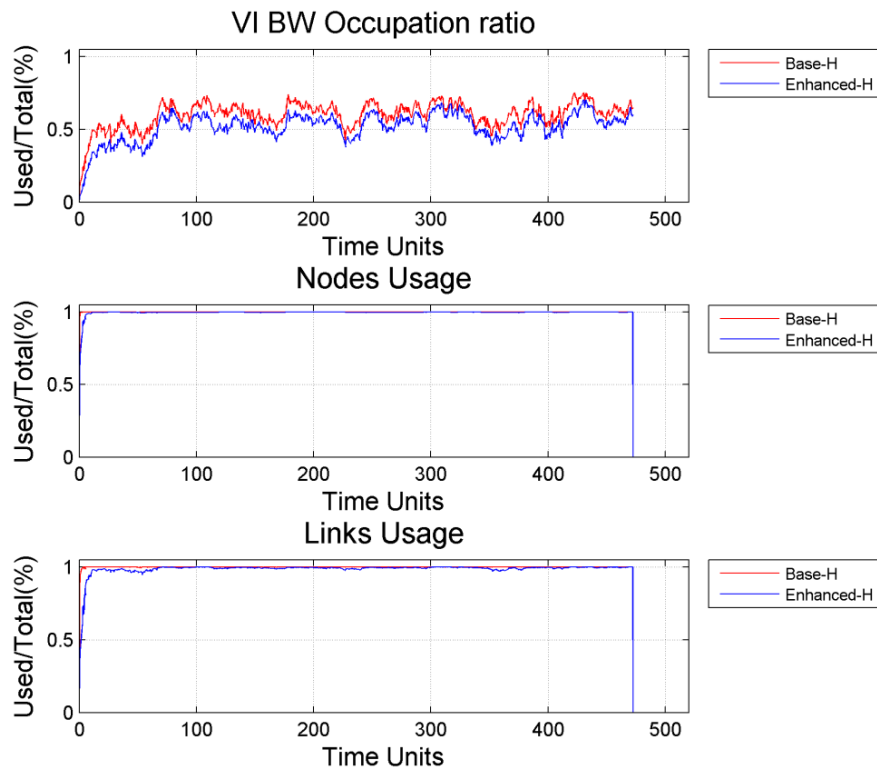


Figure 3.17 – Physical Infrastructure occupation (Substrate 45 nodes)

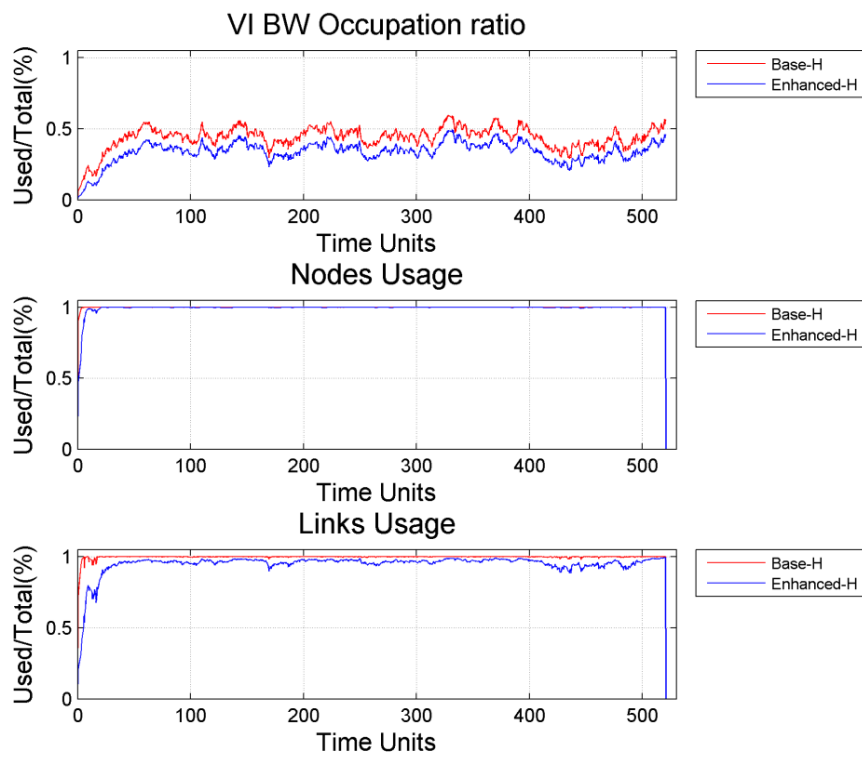


Figure 3.18 – Physical Infrastructure occupation (Substrate 55 nodes)

3.6. Conclusions

The objective of the base algorithm proposed in [62] was the load balancing of the nodes and links. The load balancing policy can be used to maximize the embedding of VIs when there are no constraints provoked by the physical infrastructure resources limitations (bandwidth of the physical infrastructure links). The use of the shortest path in an independent way, each time it is required to map a virtual link, provokes useless bandwidth occupation of the physical infrastructure, and this strategy constraints the embedding of more requests. We showed that answering directly to that main constraint, the lack of capacity of the physical connections (links), we can achieve better results. The *HOP_Dijkstra* allows the algorithm to perform better candidates quantification, because the use of links are penalized.

In our approach we strengthen the list of candidates to be more resistant to selection failures. Moreover, we assure that the conditions of work will be respected to avoid selection problems. The strategy of load balancing here was used more in a second plan, to untie between candidates upon their selection.

It is noticeable that the enhanced heuristic algorithm has better performance when considering the several metrics. Even when it presents higher VIs embedding, it also presents lower physical infrastructure occupation and power consumption. We introduced also the revenue feature to compare the quality of the VIs accepted. With this in mind, we can conclude that the approach that presents better acceptance ratio and better revenue ratio is the approach that has hosted a higher number of VIs, and those VIs are the most profitable for the physical infrastructure provider.

4. Link Reoptimization Strategy

4.1. Introduction

In this chapter we analyze the impact of a reoptimization algorithm together with the heuristic mapping algorithm approach. This chapter will describe the algorithm proposed to perform the reoptimization with the goal of reducing the bandwidth occupation of the physical infrastructure, in order to allow higher embedding of VIs into the provider infrastructure. This optimization will deal only with the virtual links and leaves the virtual nodes (where the VMs are allocated) untouched, because the consequences of remapping the virtual links are less disruptive than remapping the VMs. The process of reallocating a VM from one physical node to another is difficult, time expensive and inconvenient to perform in a large level, and a simple change of link routes is evidently preferable.

The purpose of the mapping algorithm is to find a way to host individual VI requests, set of virtual nodes and virtual links, in a physical infrastructure. The information that the mapping algorithm receives considers the properties of the requested VI and the state of the physical infrastructure in that moment. The advantage of this way of working is the simplicity of the process, because it considers only one VI at a time. The outcome of this mapping algorithm can be seen as a sum of the physical infrastructure updates that result from each successful mapping. However, this simplicity constraints the embedding of VIs, because the mapping algorithm only influences the decisions of the present VI in concern and leaves untouched the past hosted VIs. The link reoptimization tries to overcome this constraint through the analysis of a window with all the active VIs hosted in the physical infrastructure, in particular their virtual links. In the first part we present the process to save the most resources possible, then we study different strategies to influence the mapping of the upcoming VI with the goal of maximizing the embedding of VIs, in order to increase the profits of the infrastructure provider.

In section 4.2 we present a description of the developed algorithm to perform this link reoptimization followed by the pseudo-code. Section 4.3 is divided in two parts: the first part analyzes the importance of quantifying the links through their bandwidth capacity, and the second part analyzes the performance of the different approaches studied. In the previous chapter, the performance of the different modifications of the base algorithm was studied only for the online scenario of the mapping problem. In addition to that scenario, for the link

reoptimization algorithm, we consider different approaches for the same simulation moments in order to better compare between approaches. In section 4.4 it is analyzed the implications of the periodicity in the link reoptimization algorithm; this approach is also compared with the enhanced mapping algorithm presented in the previous section.

4.2. Algorithm description

The link reoptimization algorithm is located in the last part of the simulator. In other words, the simulator receives a VI request and applies the mapping algorithm. Considering the results of the mapping algorithm the physical infrastructure is or is not updated. Then, the link reoptimization is performed if the periodicity of reoptimization programmed matches the current simulator iteration.

Algorithm 2 presents the pseudo-code of the link reoptimization process, which can be divided in three different parts.

Virtual Link Ordering (lines 2-7) – The first part identifies all the links of the hosted VIs and orders them according to a specific strategy. This process is further detailed in the following section.

Virtual Link Mapping Search (lines 8-19) – This phase contains an exhaustive search to find the best way of mapping the window of virtual links (i.e. to find a solution that consumes less bandwidth resources). To amplify the success in the mapping of the sorted links, we use the *Dijkstra* approach presented in sub-section 3.4.2 without restricting the constant K to a single value. Since each mapped link will influence the next mapping, we analyze the performance of the link mapping in a global way (sum of all the resources occupation). After this exhaustive search, we save the best option in terms of global resources occupation.

Virtual Link Mapping Decision (lines 20-24) – Finally, in the cases in which the link optimization finds a solution, the third part analyzes the results and decides if it updates the substrate depending on the pre-defined strategy. From the successful cases within the search process, we can have: an improvement of resource occupation (i.e. fewer resources occupied); the same level of occupations in the physical substrate; or more resources occupied.

In the cases the link reoptimization fails in finding a solution, it is reloaded the previous condition of the substrate (lines 1 and 23).

Algorithm 2: Pseudo-code link reoptimization process

<p>input : <i>Substrate</i> (Substrate Network) , V_{Active} (Active VIs) output: <i>Substrate</i> (Substrate Network) , V_{Map} (Mapped VIs)</p> <pre> 1 SaveSubstrate(<i>Substrate</i>); 2 foreach V_{Map} v in V_{Active} do 3 foreach Link i in $v.Links$ do 4 LinksList.Add(i); 5 end 6 end </pre>
--

```

7  SortLinksList(Strategy);
8  SubstrateOccupation =  $\infty$ ;
9  while K>0 do
10     ResetSubstrate(Substrate);
11     ResetLinks ( $V_{Map}$ );
12     foreach Link i in LinksList do
13          $V_{Map}.Links(i) = FindPath(i)$  ;
14     end
15     if SubstrateOccupation> Substrate then
16         SubstrateOccupation= Substrate;
17     end
18     K = K - X;
19 end
20 if SubstrateOccupation !=  $\infty$  & DecisionPolicy then
21     Substrate = SubstrateOccupation
22 else
23     RestoreSubstrate(Substrate);
24 end

```

4.3. Strategies

The big concept behind the link reoptimization is the manipulation of the already mapped virtual links to achieve better VIs embedding. In sub-section 4.3.1, we try to achieve that goal by lowering the bandwidth occupation of the physical infrastructure, because the links capacity is known as the large constraint of the VIs embedding [62]. In sub-section 4.3.2, we compare the previous results with two different approaches.

4.3.1. Bandwidth grouping

Since the physical nodes that host the VMs of the VIs will remain untouched, because of the reasons already mentioned, we aim to assure the lowest reservation of physical links. The optimal case is the one where every virtual link occupies the lowest number of physical links. Towards that objective, we propose a strategy that takes into account the absolute paths (considering that the substrate links are all free) of the virtual links. We start by mapping the virtual links that have shorter paths to assure that these links are mapped in the best way, without reserving unnecessary bandwidth on the physical infrastructure. The following links are considered less disruptive from an occupation point of view, because they would always need to consume more links with the increasing of the link stress in the physical infrastructure. For this strategy, the decision policy will be the acceptance of the link reoptimization, if the outcome of the reconfiguration will not be worst than it was previous of the link reoptimization algorithm.

Before applying the shortest path, we can take in consideration the bandwidth capacities requested in each virtual link. In order to study that implication, we consider three ways of ordering the currently mapped virtual links:

- ***Ungrouped*** – it does not distinguish the virtual links by bandwidth. It simply orders the window of virtual links by the length of the absolute path, starting from the virtual links that present lower paths to the virtual links with higher paths.
- ***Ascending Bandwidth Grouping*** – groups are formed according to the virtual links' bandwidth (i.e. virtual links with 34Mbps form one group and virtual links with 139Mbps form another group). Then the virtual links are mapped from the group with lower bandwidth to the one with the higher bandwidth. Within each group, the *Ungrouped* ordering is used.
- ***Descending Bandwidth Grouping*** – it is identical to the ascending one; however, this one maps virtual links within groups according to a descendant way (i.e. first group contains the links with bandwidth of 139Mbps and the other group contains the links with bandwidth of 34Mbps).

4.3.1.1. Performance Evaluation

In this section we study the different grouping strategies that take into account the bandwidth capacity of the virtual links. In order to study the individual performance of the different strategies, we fix the link reoptimization to perform once in 5 iterations for the same state of the physical infrastructure (substrate nodes 15, 25, 35, 45 and 55). Table 4.1 presents the average of 10 runs with duration of 500 times units each, and a VI arrival rate of 2 per time unit (1000 VI requests per run giving a total of 10000 VI requests). Moreover, the variables K and X are set to 500 and 20 respectively. For the online scenario evaluation, presented in Table 4.2, Table 4.3 and Table 4.4, we use the same 10 runs with duration of 500 times units each, and a VI arrival rate of 2 per time unit. All values in the tables present a mean of the 10 runs with a confidence interval (CI) of 95%.

In the individual performance analysis, in Table 4.1, it is noticeable that the descending grouping form has the best results comparing with the other two, considering the average of resources saved and the success of finding a solution. About the infrastructure consumption, all strategies present similar values; however, the descending way is the one that justifies submitting those changes to the substrate. That way of ordering the virtual links has the best cost/benefit characteristic. It is important to notice that the percentage of virtual link changes and VI affected are high because we do not let any link reoptimization to modify the mapped VIs (i.e. if we let the substrate to be updated by the results of one ordering way, the immediately next link reoptimization, 5 iterations after, would have already some links already good, mapped according with that form, and the substrate would not feel so many modifications).

			Ungrouped		Ascending Bandwidth Grouping		Descending Bandwidth Grouping	
			mean	CI	mean	CI	mean	CI
Average (Mbps)	Substrate Nodes	15	463,8	49,7	1039,9	495,7	1208,5	552,7
		25	472,2	12,3	2991,6	1286,6	3727,5	1515,7
		35	457,7	19,7	5982,7	2239,7	7630,6	2488,0
		45	465,0	10,3	2647,4	1739,6	4272,8	2026,1
		55	480,2	11,6	708,3	394,1	1472,6	603,2
Failure (%)	Substrate Nodes	15	7,6	5,5	13,2	8,7	5,8	6,5
		25	14,4	10,3	23,8	12,5	7,7	7,8
		35	21,7	10,1	30,5	9,3	9,6	8,7
		45	11,1	6,4	21,4	8,1	4,4	5,0
		55	1,4	1,2	5,9	4,7	0	0
VI affected (%)	Substrate Nodes	15	60,3	15,1	60,1	15,3	60,9	14,9
		25	75,9	7,4	76,3	6,8	76,4	6,2
		35	81,7	4,0	81,4	3,9	81,0	3,3
		45	77,0	6,7	77,8	5,5	77,2	6,8
		55	72,5	4,1	72,2	4,0	72,7	4,1
VL changed (%)	Substrate Nodes	15	15,3	4,6	15,1	4,7	15,4	4,6
		25	21,0	3,4	21,0	3,3	21,3	3,1
		35	23,3	2,1	23,1	2,3	23,3	1,6
		45	18,0	3,5	17,9	3,4	18,2	3,5
		55	13,1	1,4	13,0	1,4	13,2	1,4

Table 4.1 – Individual performance - groups

For the online scenario, the aspects analyzed are the performance in terms of VI's acceptance and its correspondent revenue, in Table 4.2. The performance of the different approaches is very similar; however, it is noticeable that the descending grouping presents a slightly better performance than the ascending grouping, even for the substrate of 45 nodes in which it has lower acceptance than the ungrouped but presents higher revenue. That fact indicates that the descending grouping has hosted higher quality VIs. Furthermore, for the substrates from 15 to 35 nodes, the ungrouped way presents slightly worse acceptance and revenue, but for substrates of 45 and 55 nodes this changes, which makes us conclude that, with unlimited resources, the settling of groups after ordering the virtual links by absolute paths loses importance.

			Ungrouped		Ascending Bandwidth Grouping		Descending Bandwidth Grouping	
			mean	CI	mean	CI	mean	CI
VI Acceptance (%)	Substrate Nodes	15	21,85	6,76	21,95	6,97	22,26	7,15
		25	40,34	7,36	40,38	6,62	40,64	6,47
		35	63,33	7,04	63,17	6,86	63,69	6,85
		45	84,03	5,98	83,21	6,08	83,38	6,45
		55	92,43	8,27	92,44	8,51	92,38	8,25
VI Revenue (%)	Substrate Nodes	15	14,29	4,65	14,41	4,84	14,48	4,84
		25	32,42	7,19	32,42	6,63	32,74	6,53
		35	57,12	7,99	57,23	7,69	57,66	7,73
		45	81,04	7,18	80,21	7,34	80,27	7,51
		55	91,18	9,74	91,08	9,87	91,14	9,73

Table 4.2 – VI acceptance and revenue ratio - groups

In Table 4.3 we observe the bandwidth occupation of the physical substrate. The descending form shows a lower bandwidth consumption of the physical infrastructure for all the substrates comparing with the ascending form. For the substrates of 25 and 35 nodes, the descending form presents lower bandwidth occupation comparing with the ungroup form. It is important to notice that, besides the largest substrate (55 nodes), the ordering form that has higher acceptance ratio presents higher physical infrastructure bandwidth occupation.

			Ungrouped		Ascending Bandwidth Grouping		Descending Bandwidth Grouping	
			mean	CI	mean	CI	mean	CI
VI BW occupation (%)	Substrate Nodes	15	48,06	12,94	48,95	13,35	48,52	13,40
		25	57,25	11,21	57,61	11,63	56,86	10,93
		35	62,88	4,38	63,26	4,23	62,67	3,90
		45	51,28	5,15	51,40	5,37	50,64	4,89
		55	33,52	4,22	33,57	4,25	33,52	4,24

Table 4.3 – Physical Infrastructure link occupation ratio - groups

Table 4.4 shows the nodes and links consumption. With the exception of the largest substrate (55 nodes), it is again noticeable a slight better performance of the descending grouping form.

			Ungrouped		Ascending Bandwidth Grouping		Descending Bandwidth Grouping	
			mean	CI	mean	CI	mean	CI
Nodes Usage (%)	Substrate Nodes	15	95,76	7,53	95,72	7,53	95,58	7,53
		25	97,92	2,12	97,82	2,19	97,76	2,17
		35	99,15	0,97	99,16	1,02	99,04	1,10
		45	99,63	0,17	99,58	0,26	99,56	0,30
		55	99,26	0,57	99,27	0,53	99,31	0,49
Links Usage (%)	Substrate Nodes	15	92,47	10,41	92,45	10,47	92,29	10,36
		25	97,38	2,28	97,32	2,28	97,28	2,30
		35	99,09	0,48	99,09	0,49	99,05	0,53
		45	98,64	0,53	98,57	0,65	98,64	0,62
		55	94,98	2,30	94,87	2,37	95,31	2,21

Table 4.4 – Physical Infrastructure usage - groups

Although we conclude that there is no major difference between the studied ordering strategies for the online scenario, the direct comparison between ordering forms shows that the descending grouping ordering is undoubtedly the best. The approximated results in the VI acceptance/revenue, the physical infrastructure bandwidth consumption, the infrastructure usage and virtual link changes are caused by the randomness of the online scenario. In this sense, we apply this strategy (organize the virtual links in bandwidth groups from the higher capacity to the lower capacity) in the upcoming evaluations.

4.3.2. Different approaches

After the separation of the window of virtual links by groups of bandwidth (*Descending Bandwidth Grouping*), we now consider the ordering strategy to apply on those groups. We propose two more strategies to compare with the path strategy used in the former sub-section. All strategies use the same *Dijkstra* approach introduced in the section 3.4.2 to find the path of the virtual links.

- **Randomized** – since the big concern is the link occupation, and considering that with the group division we already gave priority to the virtual links with higher capacity, this strategy function is a simple mix of the virtual links from the old VIs in use with the newest ones. For this strategy we consider a different policy in the last part of the link optimization algorithm: it is used the solution if it improves the occupation of resources of the physical infrastructure. The key point here is to lower the stress in the physical infrastructure in order to favor the acceptance of future VIs, and to not submit changes to the physical infrastructure without being sure that we get benefits.
- **Descending Stresses** – this strategy considers the utilization of the nodes source and nodes destination that compose the virtual links. This strategy has the objective of mapping the links that have the nodes more used first in order to influence the hosting

choices of the upcoming VIs, with the aim of making it more efficient. This will intentionally stress the links between the physical nodes that host more VMs, but always map the virtual links in the shortest way possible to save physical infrastructure resources. Here it is allowed the reoptimization process to be done whatever the occupation result is, since the objective is not really to reduce the global occupation but to influence the upcoming decisions of future VIs. This last strategy uses the concept of saving resources of the physical infrastructure to distinguish within solutions in the exhaustive search.

4.3.2.1. Performance Evaluation

In the individual performance study, once in 5 iterations, we perform the different link strategies reoptimization for the same state of the physical infrastructure. Table 4.5 presents the average of 10 runs (10 different substrates) with duration of 500 times units each, and a VI arrival rate of 2 per time unit (1000 VI requests per run giving a total of 10000 VI requests). We range the size of the physical infrastructure from 15 to 55 (15, 25, 35, 45 and 55). Moreover, the variables K and X are set to 500 and 20 respectively. All values in the table present a mean of the 10 runs with a CI of 95%.

In the individual performance analysis, presented in Table 4.5, the path strategy presents the best performance with respect to the average of resources saved and to the search of a solution of mapping for all substrates. It is the only strategy that presents complete success for the reoptimization attempts in the substrate of 55 nodes. With respect to the infrastructure consumption, the path and stress strategy present similar values. However, the randomized strategy presents a significant higher value of changes in comparison with the other two strategies. For the substrate of 35 nodes, the randomized strategy presents more 5,8% of VIs affected and 1,9% of virtual links changed comparatively with the usage strategy. The values of virtual link changes and VI affected are high for the same reasons presented before.

			<i>Randomized</i>		Path		Stress	
			mean	CI	mean	CI	mean	CI
Average (Mbps)	Substrate Nodes	15	1087,1	510,6	1208,5	552,7	1086,2	557,6
		25	3464,0	1482,0	3727,5	1515,7	3534,6	1505,4
		35	7067,7	2300,5	7630,6	2488,0	7156,6	2481,5
		45	3922,6	1857,2	4272,8	2026,1	3819,7	1800,0
		55	1304,7	537,1	1472,6	603,2	1326,3	517,8
Failure (%)	Substrate Nodes	15	6,4	5,7	5,8	6,5	5,6	4,6
		25	9,3	9,1	7,7	7,8	11,7	10,3
		35	9,7	8,4	9,6	8,7	10,4	6,5
		45	7,1	8,8	4,4	5,0	9,5	10,5
		55	0,4	0,5	0	0	0,9	1,0
VI affected (%)	Substrate Nodes	15	64,1	16,2	60,9	14,9	60,5	14,5
		25	80,6	5,7	76,4	6,2	77,0	5,6
		35	86,8	3,8	81,0	3,3	80,4	4,2
		45	81,5	4,6	77,2	6,8	78,0	4,9
		55	73,0	4,1	72,7	4,1	72,7	4,1
VL changed (%)	Substrate Nodes	15	16,3	5,0	15,4	4,6	15,3	4,5
		25	22,5	3,2	21,3	3,1	21,4	3,0
		35	25,2	2,8	23,3	1,6	23,1	1,7
		45	19,3	3,5	18,2	3,5	18,3	3,3
		55	13,2	1,4	13,2	1,4	13,1	1,4

Table 4.5 – Individual performance - approaches

Afterwards, it is presented a performance study of the different link reoptimization ordering strategies for the online scenario to apply in each group resulting of the *Descending Bandwidth Grouping*. The results presented are the average of 10 runs (10 different physical substrates) with duration of 500 times units each, and a VI arrival rate of 2 per time unit (1000 VI requests per run giving a total of 10000 VI requests). Moreover, the link reoptimization is also set to be performed once in 5 iterations, and variables *K* and *X* are also set to 500 and 20 respectively. The results are presented in the tabular form. All values in the tables present a mean of the 10 runs with a CI of 95%.

For the online scenario, the aspects analyzed are the performance in terms of VI's acceptance and its correspondent revenue, in Table 4.6. The performance of the different approaches is very similar, but it is noticeable a lower performance by the *random* ordering: for the substrate of 25 nodes it has 1.23% less acceptance and 0.94% less revenue in comparison with the path strategy. For the largest substrate, 55 nodes, the *random* strategy presents the higher value of acceptance and revenue.

			Randomized		Path		Stress	
			mean	CI	mean	CI	mean	CI
VI Acceptance (%)	Substrate Nodes	15	22,00	6,89	22,26	7,15	21,69	6,68
		25	39,41	6,75	40,64	6,47	40,33	6,93
		35	63,66	7,13	63,69	6,85	64,42	6,79
		45	82,73	6,46	83,38	6,45	83,13	6,35
		55	92,74	8,24	92,38	8,25	92,47	8,31
VI Revenue (%)	Substrate Nodes	15	14,43	4,74	14,48	4,84	14,21	4,63
		25	31,80	6,82	32,74	6,53	32,63	6,94
		35	57,73	8,02	57,66	7,73	58,19	7,70
		45	79,76	7,55	80,27	7,51	80,44	7,41
		55	91,51	9,72	91,14	9,73	91,27	9,79

Table 4.6 – VI acceptance and revenue ratio - approaches

Table 4.7 shows the bandwidth occupation of the physical infrastructure. The *usage* strategy shows a slightly worse performance when comparing with the *random* and *path*. For the first case (*random* strategy), it is understandable since the *usage* strategy presents slightly better acceptance which leads to higher occupation, but in comparison with the *path* strategy it presents worse performance for the lower substrates (15 and 25 nodes) and better performance for the biggest substrates (35, 45 and 55 nodes). This can be explained since the objective of the *path* strategy is to map higher capacity virtual links as directly as possible, in order to not provoke unnecessary occupation of the physical infrastructure. However, the *usage* strategy is not completely focused to reduce the physical infrastructure occupation, but to stress in a specific order paths to affect the upcoming allocation decision. In the case the physical infrastructure resources are not a problem, the *usage* strategy reach slightly better results.

			Randomized		Path		Stress	
			mean	CI	mean	CI	mean	CI
VI BW occupation (%)	Substrate Nodes	15	48,16	13,72	48,52	13,40	48,07	13,65
		25	57,08	11,43	56,86	10,93	57,25	11,11
		35	62,88	4,69	62,67	3,90	63,02	3,97
		45	50,65	5,23	50,64	4,89	51,09	5,20
		55	33,59	4,26	33,52	4,24	33,50	4,25

Table 4.7 – Physical Infrastructure link occupation ratio - approaches

Table 4.8 shows the infrastructure consumption. Here it is noticeable a slightly worse performance from the *usage* strategy in comparison with the *path* strategy, especially for the substrates of 25, 35 and 45 nodes. The *random* strategy has the same behavior as the *path* strategy in the physical infrastructure elements usage but, as mentioned before, has a slightly worse performance in the main aspects.

			Randomized		Path		Stress	
			mean	CI	mean	CI	mean	CI
Nodes Usage (%)	Substrate Nodes	15	95,33	7,67	95,58	7,53	95,40	7,67
		25	97,86	2,26	97,76	2,17	97,86	2,12
		35	98,99	1,22	99,04	1,10	99,23	0,84
		45	99,59	0,23	99,56	0,30	99,57	0,30
		55	99,28	0,60	99,31	0,49	99,28	0,57
Links Usage (%)	Substrate Nodes	15	91,97	10,55	92,29	10,36	92,15	10,49
		25	97,39	2,33	97,28	2,30	97,39	2,19
		35	99,00	0,65	99,05	0,53	99,16	0,45
		45	98,66	0,67	98,64	0,62	98,69	0,63
		55	94,97	2,35	95,31	2,21	95,08	2,35

Table 4.8 – Physical Infrastructure usage - approaches

In summary, we conclude that there is no major difference between the studied ordering strategies for the online scenario, in spite of the fact that the direct comparison has shown the *path* strategy has better performance. We conclude that the initial division in bandwidth groups is the main aspect to take into account, and the form of the links being organized inside the groups is not much relevant. However, in the overall analysis, the *path* strategy is the one that has a more balanced performance, and in this sense, it will be used in the upcoming analysis when using the link reoptimization algorithm.

4.4. Overall Performance Evaluation

4.4.1. Periodicity

Previously, the process of link reoptimization was performed every 5 iterations. However, the periodicity of the process may have influence in the achieved results. In this sense, we perform an analysis on the periodicity over a substrate of 35 nodes. The study comprises 10 runs (10 different physical substrates) with duration of 500 times units each, and a VI arrival rate of 2 per time unit (1000 VI requests per run giving a total of 10000 VI requests) for the following variation in the periodicity of link reoptimization: every iteration, 2 iterations, 5 iterations, 20 iterations and 50 iterations. All values in the graphics present a mean of the 10 runs with a CI of 95%.

Figure 4.1 presents the results of the study in terms of acceptance, revenue ratio and substrate bandwidth occupation. We notice a small difference in the values of the three parameters between the *always*, *2in2* and *5in5* vs the *20in20* and *50in50*. In general, the higher the periodicity, the higher the gains (increased acceptance and revenue and decrease on the bandwidth occupation), with small exceptions due to the randomness factor in an online scenario.

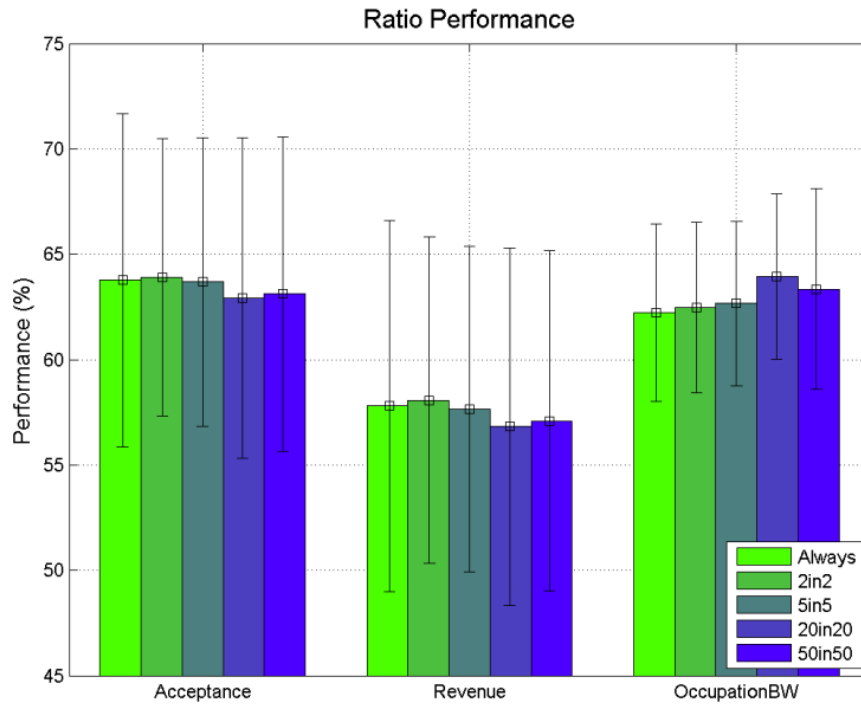


Figure 4.1 – VI acceptance and revenue ratio & Physical Infrastructure link occupation ratio - periodicity

In terms of infrastructure usage, Figure 4.2, there are no significant differences with a variation of the periodicity, since all of them use the same path finding approach (*HOP_Dijkstra*). In Figure 4.2 we also analyze the average number of changes of VIs. The number of changes increases as the periodicity increases. It is also possible to notice that there is a mean of approximately 3 virtual link changes per VI affected.

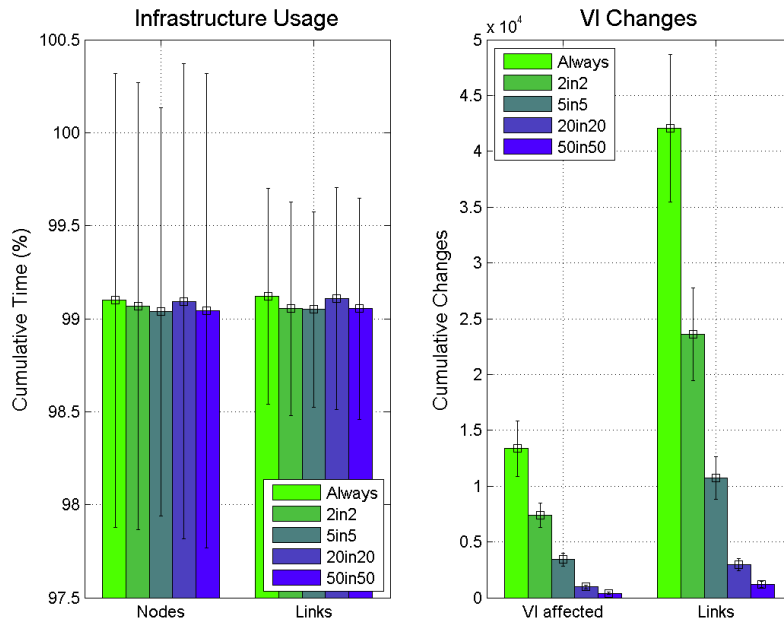


Figure 4.2 – Average number of changes on VIs & Physical Infrastructure usage - periodicity

Two inherent problems of the increase of the periodicity of the link reoptimization lay on the disruption applied to the VIs and on the simulation time. We find that the periodicity *5in5* presents a good trade-off as it slightly increases the acceptance and performance, and it does not have a high impact on VI changes compared to the *always* and the *2in2* periodicities. In the following section we consider the periodicity to be *5in5* in the heuristic with the link reoptimization.

4.4.2. Heuristic comparison

In this sub-section we study the performance of the base heuristic algorithm [62], enhanced heuristic algorithm and the enhanced heuristic algorithm with link reoptimization algorithm programmed to perform once per 5 iterations. We study the performance of the algorithm ranging the size of the physical infrastructure from 15 to 55 nodes (15, 25, 35, 45 and 55). Moreover, we fix the infrastructure size and vary the VI arrival rate from 1 to 5 (1, 2, 3, 4 and 5). The analysis comprises 10 runs (10 different substrates) with 500 time units with a VI arrival rate of 2 per time unit (1000 VI arrivals per run with a total of 10000 VI arrivals). The study focuses on the analysis of the acceptance ratio, revenue ratio, occupation of bandwidth of the substrate and node/link usages. All values in the graphics present a mean of the 10 runs with a CI of 95%.

Figure 4.3 and Figure 4.4 show the comparative results when varying the substrate size and VI arrival rate. In both cases there is a clear improvement from the base heuristic (*Base-H*) to the other two. The enhanced heuristic with link reoptimization (*Enhanced-H-LR*) is the one presenting a slightly better performance compared to the one without link reoptimization (*Enhanced-H*) in terms of acceptance and revenue. This difference is clearer for a substrate size of 35 nodes. We also observe that, for the substrate of 55, all algorithms present similar results: with unlimited substrate capacity (number and capacity of nodes and links), the load balance strategy used in the base heuristic [62] has the potential to have the same or better results, because it does not overload specific links when compared to the *HOP-Dijkstra* approach performed in the enhanced heuristic with link reoptimization and without link reoptimization.

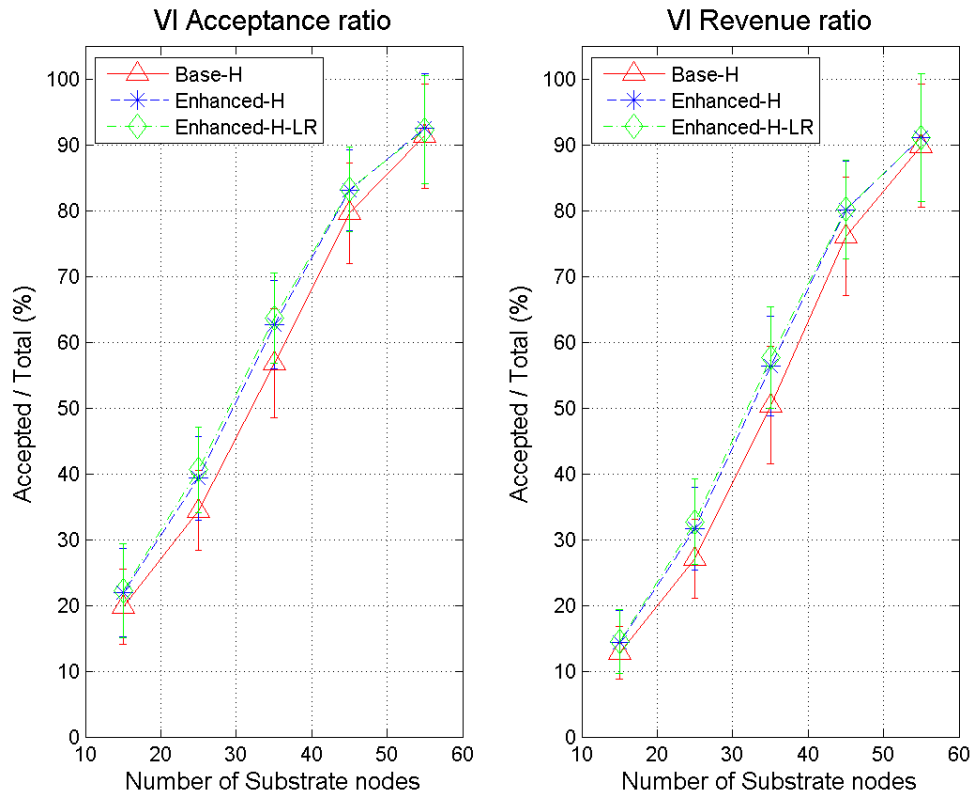


Figure 4.3 – VI acceptance and revenue ratio

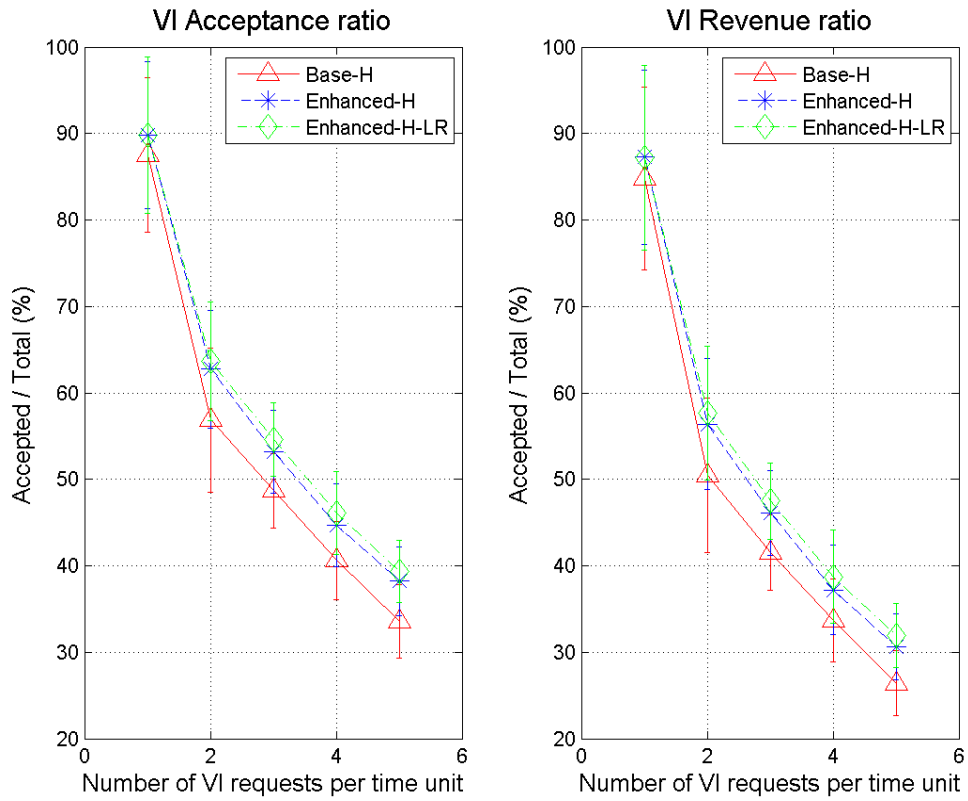


Figure 4.4 – VI acceptance and revenue ratio - request rate variation

Figure 4.5 shows the *Enhanced-H-LR* improvement in the bandwidth occupation of the physical substrate. Improvements go up to near 2.5% for a substrate of 35 nodes. Assuming again the case of unlimited substrate resources, the performance of the *Enhanced-H-LR* will always be better because of the better policy implied. The results indicate that there will always be a gap between the *HOP-Dijkstra* based approaches and the *Base-H* independently of the size of the physical infrastructure.



Figure 4.5 – Physical Infrastructure link occupation

Figure 4.6 and Figure 4.7 show that the utilization of nodes and links of the physical substrate is nearly the same for the *Enhanced-H-LR* and the *Enhanced-H*. However, the former seems to reduce slightly the infrastructure power consumption. Moreover, there is a clear difference between these latter compared to the *Base-H*. This is easy to understand because the *Base-H* does not use the *HOP-Dijkstra* approach.

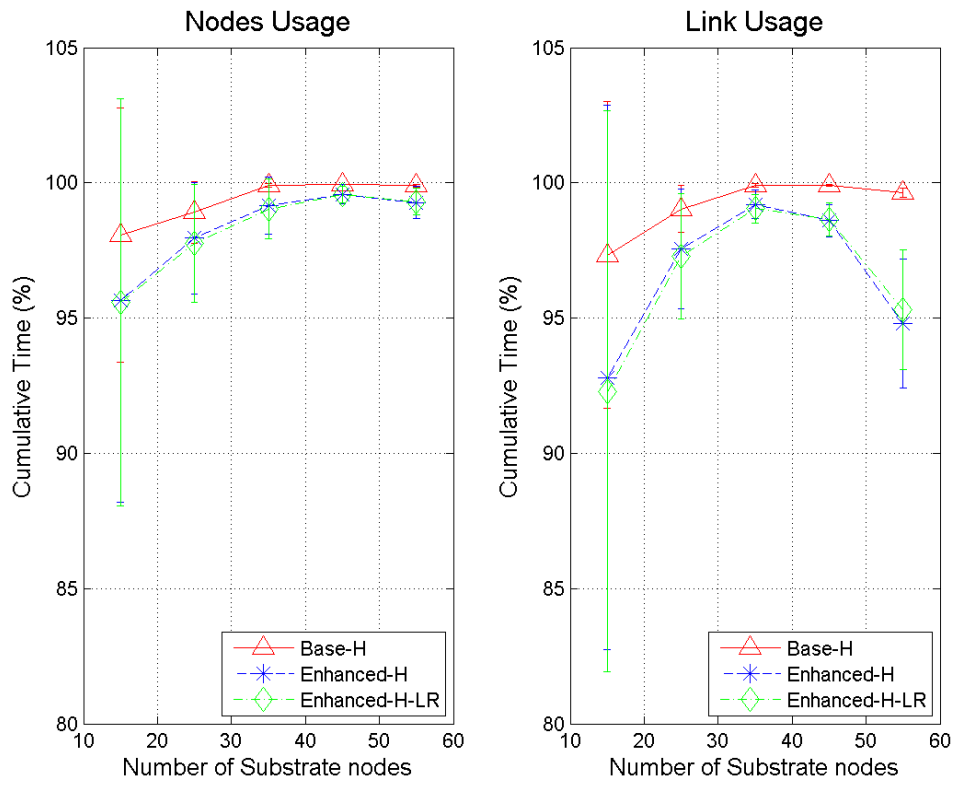


Figure 4.6 – Physical Infrastructure usage

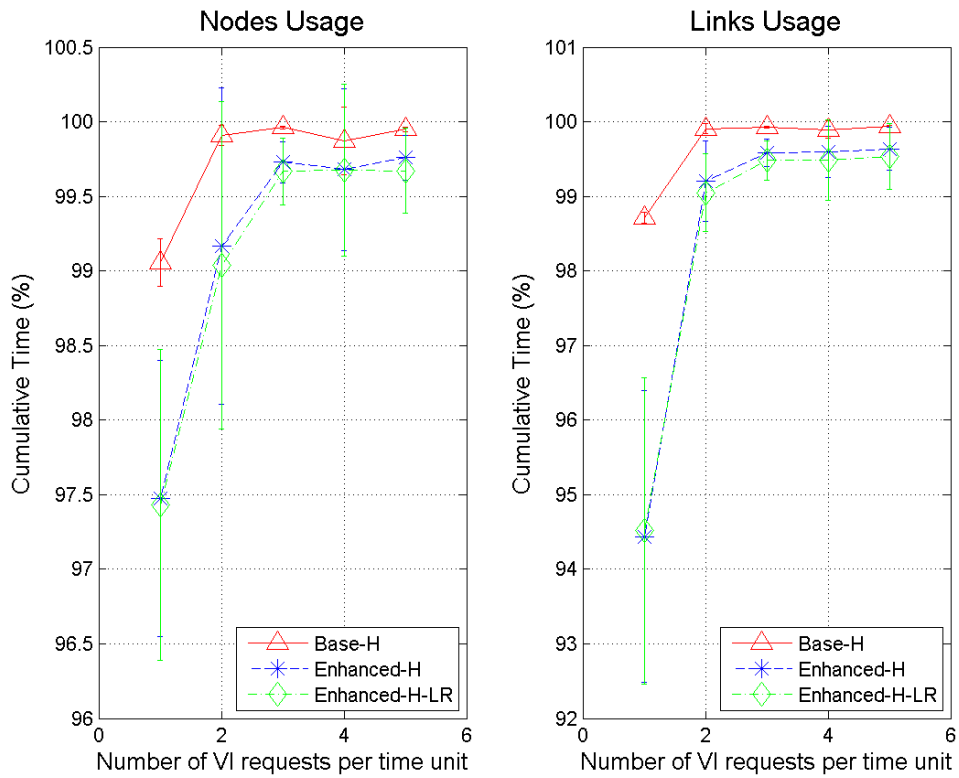


Figure 4.7 – Physical Infrastructure usage - request rate variation

In summary, the best performance of the *Enhanced-H* compared to the *Base-H* is clear in all perspectives (i.e. acceptance, revenue, bandwidth occupation and physical infrastructure usage). As for the *Enhanced-H-LR* compared to the *Enhanced-H*, the difference is not as significant, but still there are some improvements. The improvements are clearer in the case of VI arrival rate variation, as observed in Figure 4.7.

4.5. Conclusions

This chapter described the link reoptimization approaches to obtain the best embedding of VIs and get the best revenue possible. The main conclusion obtained was that the definition of link groups by bandwidth, starting from the links with higher bandwidth capacity to the ones with lower bandwidth capacity, presents the best results from a profit point of view to the infrastructure provider (more acceptance and lower power consumptions). Associated with this group ordering, if the *HOP-Dijkstra* approach is used in the mapping of the virtual links, we concluded that the order applied inside those groups is not relevant.

The aspect that the link reoptimization most improved was the bandwidth occupation of the physical infrastructure. That was the objective of the link reoptimization in order to tackle the biggest limitation for the embedding of VI. However, the embedding of VI did not suffer the same improvement that the bandwidth occupation experienced. This dissociation with both results can be the result of the path finding algorithm used. In this chapter, we leave an open point to future study. In our opinion, the structure of the link reoptimization is well designed, but the *HOP-Dijkstra* path finding algorithm might be less appropriated to be applied, and with that constraint, it may reduce the success of the link reoptimization.

5. Software-Defined-Network oriented Service Manager

5.1. Introduction

The SDN techniques have the objective to decouple the control operations from the forwarding elements. In the light of that simplification, the forwarding elements operate in the identification of packets and the posterior matching with pre-established rules. This chapter proposes a module to our SDN framework to enable the reception and treatment of connectivity services that are further translated into rules to apply in the substrate network elements. Moreover, this module, the Service Manager, is responsible to maintain the good operation of those services in the substrate network. We present our SDN framework architecture in the section 5.2 with an overview of all the modules in which the *Service Manager* is integrated. In section 5.3 it will be presented the architecture of this module in detail. Furthermore, in section 5.4 it is presented the tests of performance of the module independently, and then integrated in the complete solution for the activation of connectivity services. The SDN overall architecture has been developed in the framework of two MSc Dissertations.

In the previous chapters 3 and 4, it was showed that the capacity of the connections (links) limits the good operation of the substrate network. Taking those conclusions in consideration, we will use the normal *Dijkstra* algorithm to obtain the shortest path to map the flows. Since the SDN framework is at an early stage and the substrate network is small and simple, this path finding strategy is appropriate to resolve the path finding problem without jeopardizing the good operation of the substrate network.

5.2. SDN Framework Architecture

The architecture of our SDN framework is presented in Figure 5.1. The SDN framework is composed by one module in the application plane and four modules in the control plane. Each module has independent functions and all together form the basic cycle of our SDN solution to enable connectivity services. With all these elements, it is allowed to receive, treat, manage and respond to network problems that are related with those services. Note that the *Service Manager* module that is proposed by this Dissertation is the group of one module from the application

plane, *Service Handler*, and one module from the control plane, *Flow Handler*. This module is separated in two sub-modules because, in the architecture point of view, the *Service Manager* encompasses functions in the applications plane and functions in the control plane. In terms of implementation, these functions are presented together in the same module as detailed in the following section. Following we give an overview of all the modules.

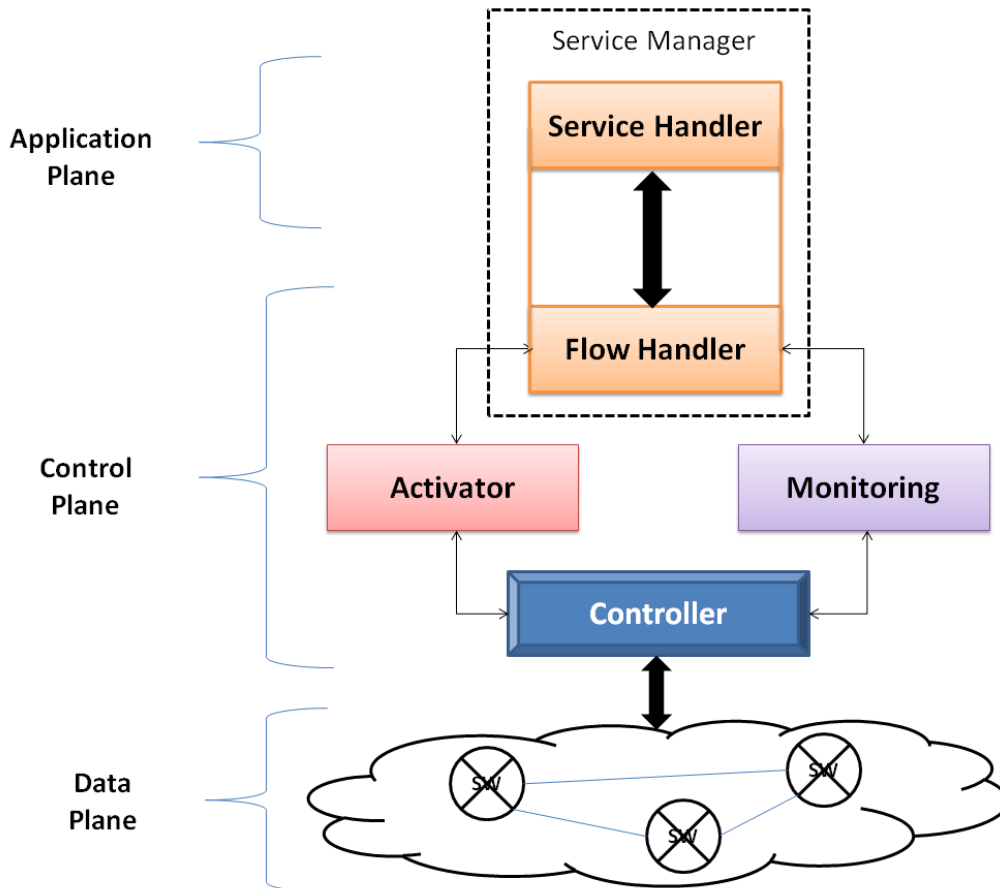


Figure 5.1 – SDN Framework Architecture

- **Service Handler** – this module has a RESTful API, which receives the connectivity services requests to activate. *Service Handler* has the vision of connectivity services and flows. Its objectives are the translation and storage of the resulting flows correspondent to the connectivity services requests. This module is also responsible for indicating the state of the connectivity requests.
- **Flow Handler** – this module has the vision of flows and rules. *Flow Handler* is responsible to activate the flows resulting from the translation of connectivity services. In order to do it, it gathers the information of the network (with the use of a RESTful API) and decides the network links (paths) in which the flows will traverse (with the use of a *Dijkstra* algorithm). Besides that functionality, this module manages the flows in terms of optimizations and in terms of problems that may exist in the substrate network.

The following modules are external of this Dissertation work, developed in a parallel Dissertation, but will be explained to be easier to understand the options made in the development of the *Service Manager*.

- **Activator** – this module is responsible to create the set of rules to implement in the forwarding elements via the *Controller*. Besides the rule information, it needs the indication of the position of the hosts (MAC address and port of the switches which are connected) that will connect, and the network links which the connection affects, in the form of an adjacency matrix. This matrix indicates the elements of the network that are needed to insert the rule in their flow table.
- **Monitoring** – this module is responsible to retrieve and treat information regarding the data plane via *Controller*. This works in a subscription system. The modules that are interested in receiving the information about the data plane need to perform a subscription, and after that they are notified (receive the updated information) when modifications and/or triggers exist in the substrate network (i.e. alterations of topology and alarms).
- **Controller** – this is the entity responsible for the communications between the control plane and the data plane. In our solution, it is used the *Floodlight* controller that uses the *OpenFlow* protocol to identify and manage the forwarding elements of the substrate network. The previous two elements contact directly with this module to retrieve information and modify rules of the substrate network elements.

In summary, the *Service Manager* subscribes the services of topology and alarms from the *Monitoring* module in order to receive the current substrate topology, and be able to receive the triggers that can be sensed in the substrate network. In the event that the *Service Manager* is requested to activate a connectivity service, it uses the network information to build the paths of the flows. After that, this module activates them using the module *Activator*. If everything works well, after that point the connectivity service is considered active. The following section presents the *Service Manager* in detail.

5.3. *Service Manager* Implementation

The *Service Manager* is developed using the programming language python, and requires the installation of some python libraries and the data base management system MySQL. The python libraries that this module uses among its functions are: *bottle* (is the Web Server Gateway Interface (WSGI) used to put the *Service Manager* online); *httplib* (used to exchange data through the HTTP protocol between modules, it is the supported protocol of our SDN framework); *json_schema_validator* (used to analyze the form of the input parameters for consistence matters); *json* (is the format in which the data is passed between modules); *pickle* (save the objects information into internal files); *networkx* (used to find the path for the flows); and *MySQLdb* (used to interact with the database MySQL).

In the following sub-sections we will present the functions of each sub-module that forms the *Service Manager*, in Figure 5.2. The sub-modules that correspond to the *Service Handler* in the SDN architecture are the top *REST API* and the *Service Receiver*. Its actions will be directly reflected in the database. The following sub-module corresponds to the *Flow Handler* in the SDN architecture, which will get the flows from the database to assign the path and activate them. A flow is considered completed if it is a network path (topology) associated and is ready to be activated (function of the *Flow Activator*). This part performs also the subscription demanded by the *Monitoring* block in order to receive the information of the substrate (switches and links in the form of an adjacency matrix) and to be enabled to receive eventual problems in the substrate network. Furthermore, this part is responsible to maintain the good operation of the connectivity services and correspondent substrate network.

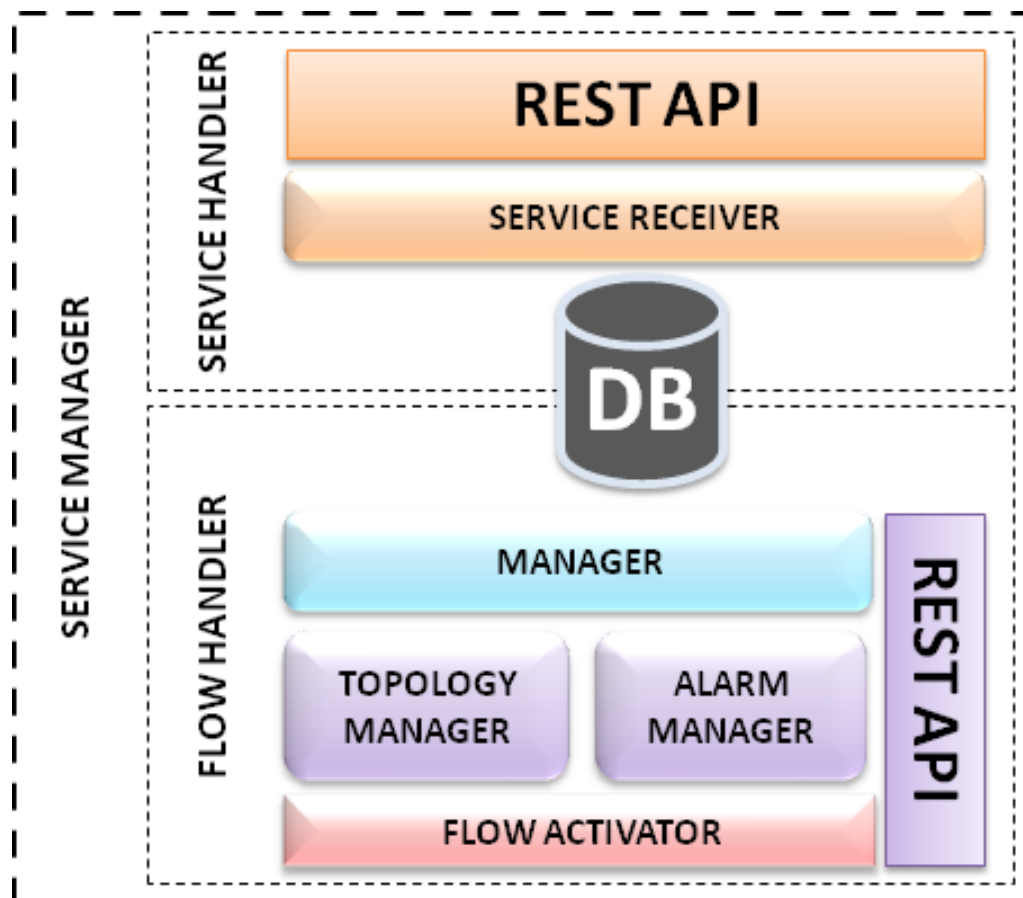


Figure 5.2 – Service Manager architecture

5.3.1. Service Handler

The applications that are interested in activating connectivity services have to use the method *POST* from the *RESTful* principle through the HTTP protocol for the Uniform Resource Locator (URL) presented in Table 5.1. The *ip:port* needs to be the same defined upon the launching of the server with the bottle WSGI. Moreover, the requester needs to send the information of the connectivity service according to the format *json*. If all the mandatory fields of the service request are presented, it is translated the connectivity service into flows and storage in the database.

Then, an ID number is sent in an answer for the requester, in order to know how to later locate that connectivity service. The ID number is used to obtain the state of the connectivity service (method *GET*), to perform an update in those request elements (method *PUT*) and to deactivate the request (method *Delete*), in Table 5.2.

URL	http://ip:port/mapping
GET	X
POST	Receives a connectivity service to activate and send back its ID from the database where it was saved
PUT	X
DELETE	X

Table 5.1 – URL to receive the connectivity services to activate

URL	http://ip:port/mapping/<request_id>
GET	Sends the information of the connectivity service where the ID is the request_id (data inside the database)
POST	X
PUT	Receives an update for the connectivity service where the ID is the request_id (data inside the database)
DELETE	Deletes the connectivity service where the ID is the request_id (data inside the database)

Table 5.2 – URL to interact with the created connectivity services

Upon the creation of the connectivity service, the requester needs to send information according with a formulation stipulated by this module. It sends an object called “*method*” that contains two features: a string called “*name*” in which it will be mentioned the way of working of the service, and an array called “*elements*” in which it will be mentioned the hosts correspondent to this service. In addition to the previous object, it is included another object called “*rule*” that is composed by two strings: one called “*name*” that identifies the type of the connectivity service, and another called “*subName*” that imposes a condition to apply with the previous type (this field is optional). As an example:

```
{“method”: {“name”: “...”; “elements”: [...]; “rule”: {“name”: “...”; “subName”: “...”}}
```

Currently, two types of work are supported: full-mesh (“fullMesh”) and multicast (“hub”). For the full-mesh form, the result will be a bidirectional connection between each two elements inside the request. With the full-mesh work mode the service will be translated into combinations of all elements two in two flows (i.e. with two elements we only have two flows, but with three elements we already have six flows). For the multicast working form, the service requester needs to identify which element is source and which elements are targets of the connectivity service. This type of work will create equal number of flows with the number of target hosts where the data is only allowed to flow from the source host to the targets hosts. The type of work field is a mandatory field.

The hosts have two mandatory fields and four optional ones. The mandatory fields are the “swMac” and the “swPort”, that is a string containing the MAC address and a number containing

the port of the switch which is connected, respectively. The optional fields are the “IP”, “MAC”, both are of string type, and “hub”, “target” that are of boolean type. These optionally fields become mandatory if inside the rule it is presented the IP and/or MAC (“IP” and “MAC”) indication, or if the multicast ways of work are chosen (“hub” and “target”).

The rule expresses one advantage of the SDN solutions by allowing to work in the layer 2 (MAC) or layer 3 (IP) or even with both. Currently, it is supported the following options: for the “name”: PORT, IP, MAC, IP_MAC, UDP, TCP and ICMP; for the “subName”: IP, MAC and IP_MAC.

If the previous restrictions are respected, the individual flows of each request are stored into the database. It is important to notice that the *Service Receiver* is protected against the elements that are connected in the same port of the same switch (i.e. hosts coming from the same cloud). In those cases no action will result between those elements. Moreover, if a single aspect is not respected, the flows that were built from that service are removed from the database; the rest of the flows are dropped and an error message is sent.

5.3.2. Database

In this sub-section it is presented the database tables with all the items of the connectivity service requests, which are stored by the *Service Receiver*, Figure 5.3.

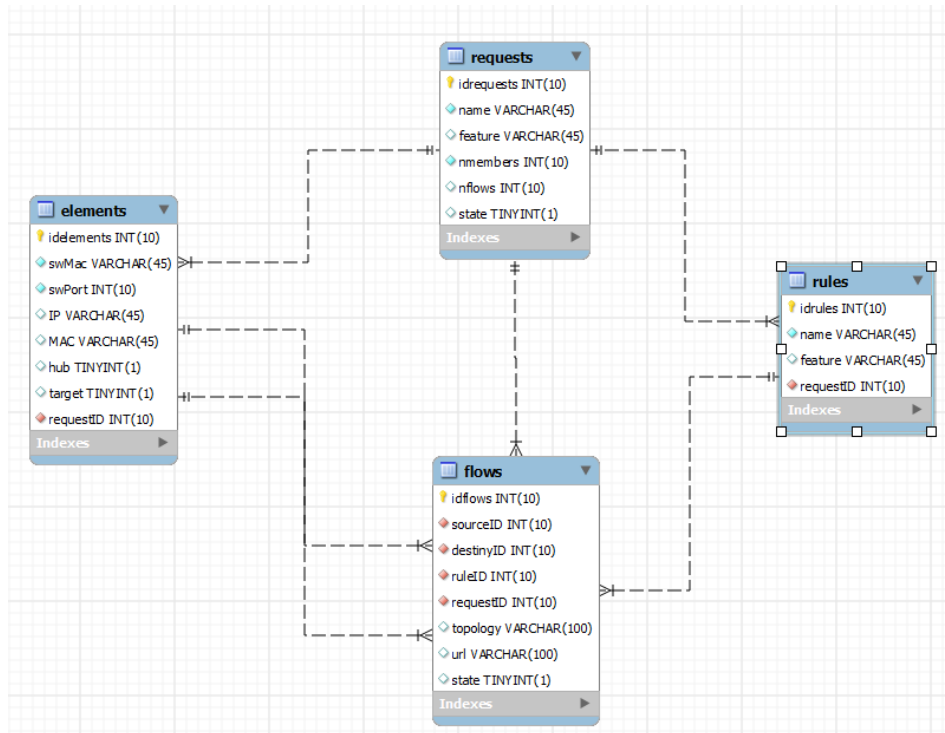


Figure 5.3 – Service Manager Database

The database consists in four tables:

- **Requests** – this table registers the form of work chosen by the service into the field “name”. The field “feature” saves the type of connectivity service that was requested (i.e.

UDP). In addition, it is also presented the number of elements that the service has, and the number of flows that were translated into fields “nmembers” and “nflows”, respectively. At last, the table gives information of the state of activation of the service, in the field “state”. If every flow is active the connectivity service is considered active and this field has the value 1. All this information is associated with a unique identifier which is represented by the field “idrequests”. Upon the acceptance of the connectivity service request, this is the field returned to the requester (service ID inside the database).

- **Rules** – this table saves the connectivity information of the services. This uses a foreign key to relate that information with a specific connectivity service, field “requestID”. The field “name” saves the type of connectivity of the rule, and the field “feature” saves the subtype of the rule that was requested to be applied with. This represents the “subName” mentioned in the previous sub-section.
- **Elements** – in this table it is saved all the information of every element (host) that the connectivity service wants to be applied. It also uses the foreign key to relate those elements with a specific service, field “requestID”.
- **Flows** – the *Service Receiver* will save the flows translated from the connectivity service into this table. The initial part of every flow is composed by a source host (“sourceID”), destination host (“destinyID”) and a rule (“ruleID”). All these fields together with the “requestID” are foreign keys pointing to the correspondent elements of the previous tables. Furthermore, when a connectivity path is allocated, the field “topology” is updated with the address of the internal data structure that contains the network path information. Every flow also contains a field “url” and a field “state”. The first field is used to save the URL address where it is possible to check the aspects of activation provided by the *Activator* module, and the second field has the objective to inform if the flow is activated, if it has value 1, or is deactivated, if it has value 0.

Associated with the tables, the database management system MySQL allows to program triggers. We use these triggers to update automatically the number of elements, number of flows and the state of the requests. When any elements/flow is added, the database increases 1 to the field of the respective table requests. The same happens when an element/flow is eliminated. Together with that, the database also reacts when any modifications are performed in the flows table (i.e. addition of flows, alteration of flows and elimination of flows). The database will check the state of all the flows from the affected connectivity service and updates, if necessary, the state of that service in the table “requests”.

5.3.3. Flow Handler

The module *Monitoring* works with a subscription system. Since we are interested on some of its services, it was needed to create two dedicated URLs to receive information when the substrate network is modified and when some concerns regarding the operation of the substrate network is found, Table 5.3 and Table 5.4 respectively.

URL	http://ip:port/topology
GET	X
POST	X
PUT	Receives the information of the substrate network
DELETE	X

Table 5.3 – URL to receive the substrate network information

URL	http://ip:port/alarms
GET	X
POST	Receives the information of the triggered alarm.
PUT	X
DELETE	X

Table 5.4 – URL to receive the alarms regarding the operation of the substrate network

The sub-module *Topology Manager* is responsible to treat the information of the substrate network, Table 5.3. With the indications of the switches MAC addresses of the substrate network, the adjacency matrix of the connectivity and their capacity between those elements, it updates the internal file that is reserved to save that information. However, it does not delete the previous aspects of the substrate network. Then, the *Topology Manager* signals the sub-module *Manager* that there is a change of topology (for the initial moment it only means that it is already available information of the substrate network). The sub-module *Alarm Manager* does the same thing as the previous sub-module, but with the information regarding only one switch. The alarm identifies the port of the switch, so it gives the MAC address of that switch with its affected port, and the MAC address of the other switch with its correspondent port that has the affected connection. Then, this sub-module also signals the *Manager*, in this case with information on the location of the problem.

The first function of the sub-module *Manager* is to register the dedicated URLs into the correspondent services provided by the module *Monitoring*. After the subscription, the *Manager* (and the signal from the topology manager) waits for the arrival of connectivity services. In the time that flows are stored into the flows table of the database, this sub-module tries to find a path to connect the source host with the destination host, saving that information into an internal data file. The address of that internal data file is inserted into the field “topology” of the flow table inside the database. This works in this way because the size of the path information is dynamic (the size of the matrix increase with the number of elements of the substrate network), and the fields inside the database should have a fixed size. More than allocating a topology to the flows that are inserted in that time instant, it is always checking if there are flows that have this field empty in order to try to complete them. The last part of the initial cycle is the activation of the flows that already have a topology with the path associated. This sub-module is always looking for flows in those conditions to activate them through the sub-module *Flow Activator*. When another signal from the *Topology Manager* is received, this sub-module first identifies the differences in the substrate network (i.e. if the network lost and/or got elements). For the cases that the substrate network loses elements, this sub-module analyzes if some of the flows with network path allocated were affected. In affirmative cases, it tries to overtake that problem by finding another path to connect the hosts. In the other hand, when there is an addition of

elements, this sub-module tries to optimize the path of all flows with already network paths allocated. Since the objective is to connect the hosts by the shortest path, in the case that it is found a better path in comparison with the previous given, it is updated through the *Flow Activator*. When a signal from the *Alarm Manager* is received, the *Manager* locates the affected flows and tries to overtake that problem, similarly with the case that the substrate network has lost elements.

The sub-module *Flow Activator* receives the identification of the flows and the action that will be performed: activate or update. This is the sub-module that performs the communication with the module *Activator*. From those communications, the *Flow Activator* receives the URL in which those flows can be checked. This sub-module includes that address into the field “url” in the table flows of the database. The module *Activator* gives a unique URL to every unique flow request. It is important to mention that this sub-module is protected and does not delete flows from the substrate network without being sure that the flow is not shared by multiple services.

5.4. Performance Evaluation

A single connectivity service can be translated into several flows. We will study the performance of the *Service Receiver* for the different form of work (full mesh and multicast) and of the *Manager* for assigning the path for the resulting flows. The performance is tested in a machine with the features presented in Table 5.5.

CPU Model	CPU Freq.	CPU Cores	CPU Threads	HDD Memory	RAM Amount	RAM Freq.
Intel Core 2 Duo E6400	2.13 GHz	2	2	145 GB	4 GB	533 MHz DDR2

Table 5.5 – Physical machine features

For the full mesh working system, we changed the number of hosts between 2, 3, 5, 7, 10 and 20. In the case of the multicast working system, we added a connectivity service with 50 hosts in addition to the previous ones. The machine where the host source is allocated is different from the machine where the rest of the hosts are allocated. The analysis comprises 10 runs, always starting with the database empty. The study focuses on the analysis of the time consumption of each sub-module. All values in the graphics present a mean of the 10 runs with a CI of 95%.

Figure 5.4 shows that the time consumption of the full mesh form has exponential behavior, while the multicast form, Figure 5.5, it has linear behavior with the increment of the number of hosts. This is expected, since the full mesh work system creates two flows between each two hosts. As mentioned before, it is a combination of the number of hosts two in two which results in a much higher number of flows comparing with the multicast form for the same number of elements. We can conclude that the creation of flows is the process that takes more time inside the *Service Receiver*, and that time is directly proportional to the number of flows translated. Besides that, the initial tables’ allocation (table request element and rules) also consumes some time, but that became irrelevant with the increasing of the number of resulting flows. For the multicast working form, each flow needs approximately 0,095 seconds to be formed and saved

into the database; meanwhile, for the full mesh, it needs approximately 0,05 seconds per flow. This is easy to understand because, after calculating the connection attributes, it is only needed to change directly the source and destination attributes to obtain the other way connection (full mesh creates a bidirectional connection between the elements).

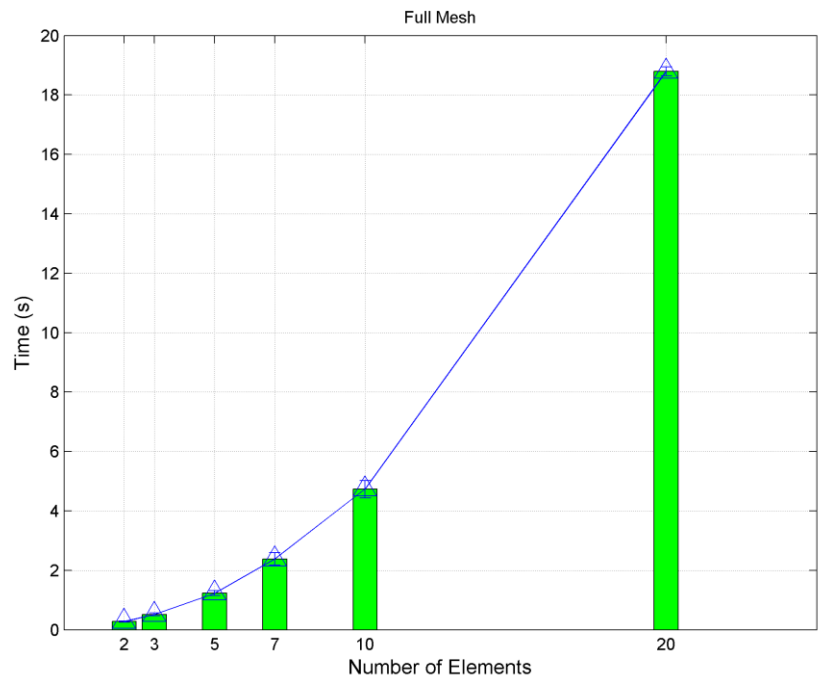


Figure 5.4 – Service Receiver performance Full Mesh

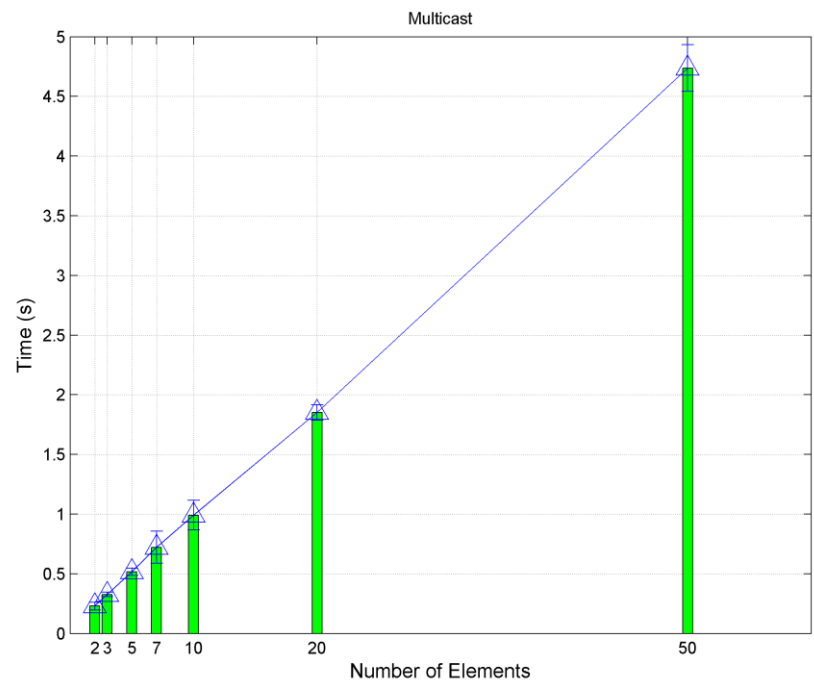


Figure 5.5 – Service Receiver performance Multicast

In Figure 5.6 and Figure 5.7 we observe the time consumption for the path assignment performed by the *Manager* in addition to the storage and treatment of the connectivity services performed by the *Service Receiver*, for both ways of work (full mesh and multicast). Furthermore, it is presented the total time needed from the initial moment until the flows are ready to be activated. The performance of the sub-module *Manager* for path assignment presents linear time consumption. Each path assignment takes approximately 0,055 seconds to be built, stored inside an internal file, and updates the corresponding flow in the database independently of the number of flows.

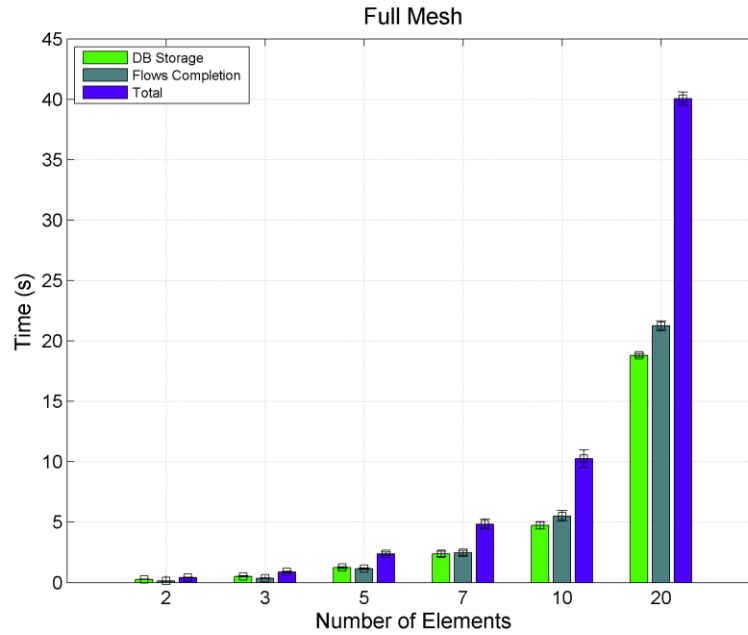


Figure 5.6 – Time consumption for the translation, storage and completion of the flows (Full Mesh)

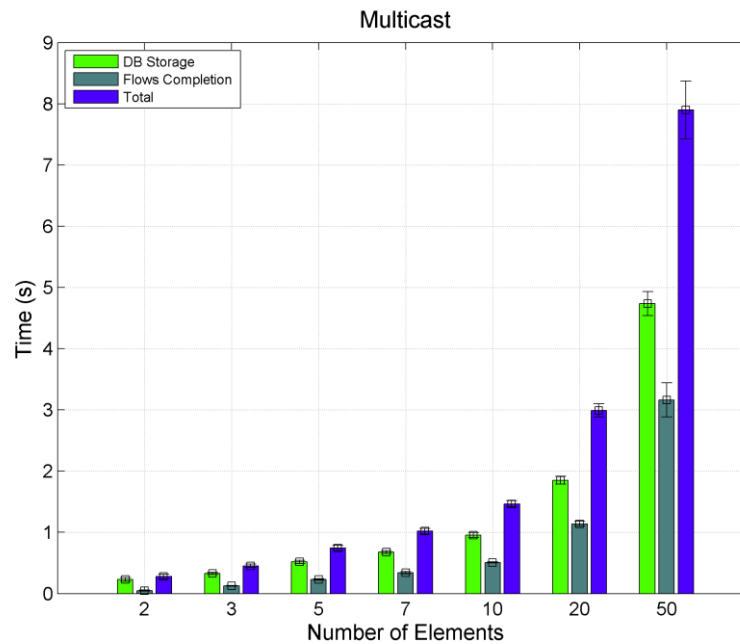


Figure 5.7 – Time consumption for the translation, storage and completion of the flows (Multicast)

Now we will study the performance of our complete SDN framework solution. Due to limitations with the available hardware, our substrate network is composed by 4 switches (A, B, C and D), and their characteristics are shown in Table 5.6. In the following tests, we fix the connectivity service to work in the multicast operation form and we changed the number of elements (2, 5 and 20). Moreover, we study the response of our SDN framework when the substrate network suffers changes, loss of a link b) and addition of a link c), in Figure 5.8.

NODE	CPU Model	CPU Freq.	CPU Cores	CPU Threads	HDD Memory	RAM Amount	RAM Freq.
A	Intel PentiumD 950	3.40 GHz	2	4	40 GB	6 GB	667 MHz DDR2
B	Intel Core 2 Duo E6400	2.13 GHz	2	2	145 GB	4 GB	533 MHz DDR2
C	Intel Core 2 Duo E6400	2.13 GHz	2	2	145 GB	4 GB	533 MHz DDR2
D	Intel PentiumD 950	3.40 GHz	2	4	40 GB	6 GB	667 MHz DDR2

Table 5.6 – Testbed specification

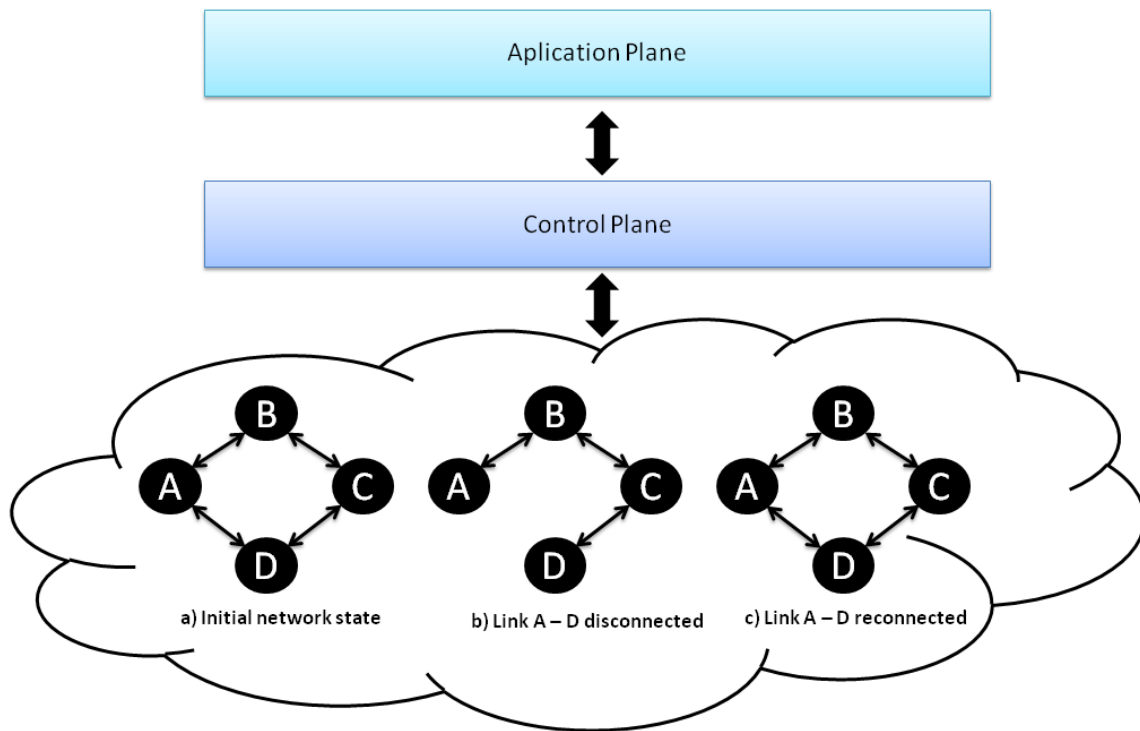


Figure 5.8 – Substrate Network moments of performance evaluation

The host source will be always placed at the node A. The targets will be placed: 1 at node D (service with 2 elements in which it will be translated into 1 flow); 2 at node B and 2 at node D (service with 5 elements which will be translated into 4 flows); and 3 at node A, 4 at node B and 12 at node D (service with 20 elements which will be translated into 19 flows). First, the study focuses on the analysis of the time that the initial connectivity service takes from its origin until it is activated, Figure 5.9. After the substrate network is changed, it is studied the time that our SDN framework takes from that moment until it is resolved (if service is affected, its flows will be

optimized for the service to remain active), Figure 5.10 and Figure 5.11. All values in the graphics present a mean of the 10 runs with a CI of 95%.

The complete SDN framework tests are presented in the last three figures, Figure 5.9 Figure 5.10 and Figure 5.11. In Figure 5.9, it is noticeable that the services that have more flows present better average time per flows in the storage. As mentioned before, the allocation of the other tables (besides the flow table) does not present strong time changes with the increasing of elements, therefore the time per flow decreases. The path assignment presents a linear behavior. Since it has a sequential behavior (assigns the path for one flow at the time), the only aspect that matters is the number of flows that are incomplete in that moment. The services that resolve in a higher number of flows present higher time consumption for the path assignment. The activation time presents also a linear behavior. The *Flow Activator* receives one flow at the time. It takes approximately 0,140 seconds to activate each flow.

Figure 5.10 and Figure 5.11 present the time that our framework needs to react to the change of topology, b) and afterwards c) at Figure 5.8. The time consumption that identifies the problem and changes the topology in the internal data structure is similar for the three experiments (2, 5 and 20 elements), since it is a process independent of the number of elements. The initial step of that reaction is the perception that there was a change in the substrate network by the *Monitoring*, that same module obtains the resulting substrate network aspects and communicates them to the modules that are subscribed (*Service Manager*). The last step is the update of the internal data file with the substrate aspects performed by the *Service Manager*. It is noticed a large time difference between the reaction for the lost of a link, approximately 0.1 seconds, and the reaction to the addition of a link, approximately 3 seconds. For the first case, the reoptimization time will be directly proportional to the flows affected, because it is needed to find another path to allocate them activated again. For the last case, it is performed a path finding attempt for every flow. In our experiments, the time for both cases is almost the same, because the number of flows that get affected with the loss of a link is the same that after getting a shorter path with the addition of that same link. In addition, we noticed in our experiments that the search for the flows affected when there is a loss of a link consume more time than a path finding try for the flows that will not be changed.

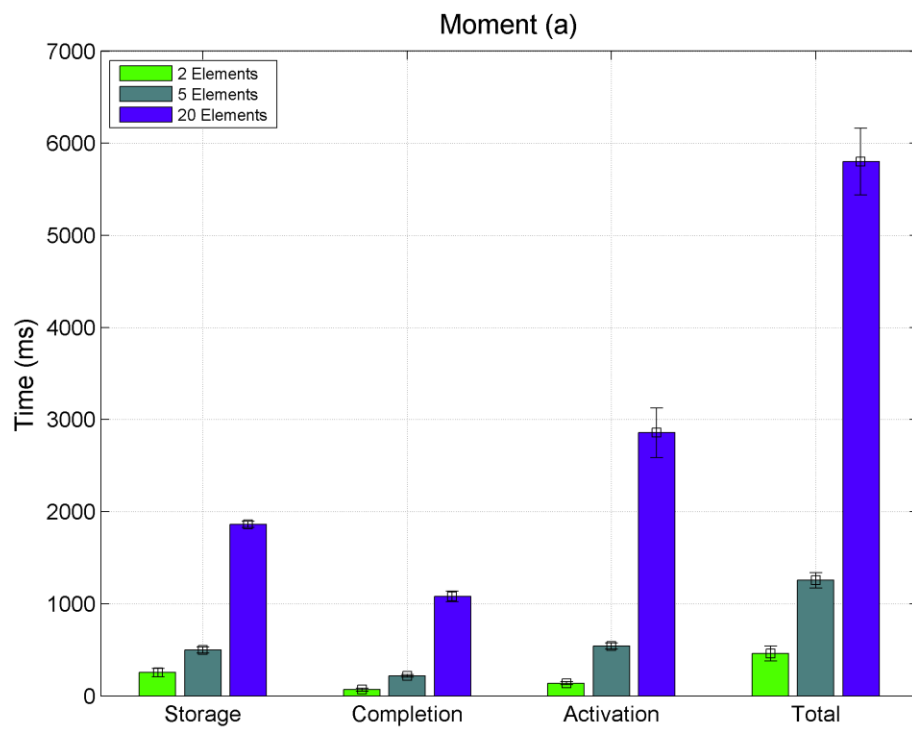


Figure 5.9 – Time to activate the connectivity services

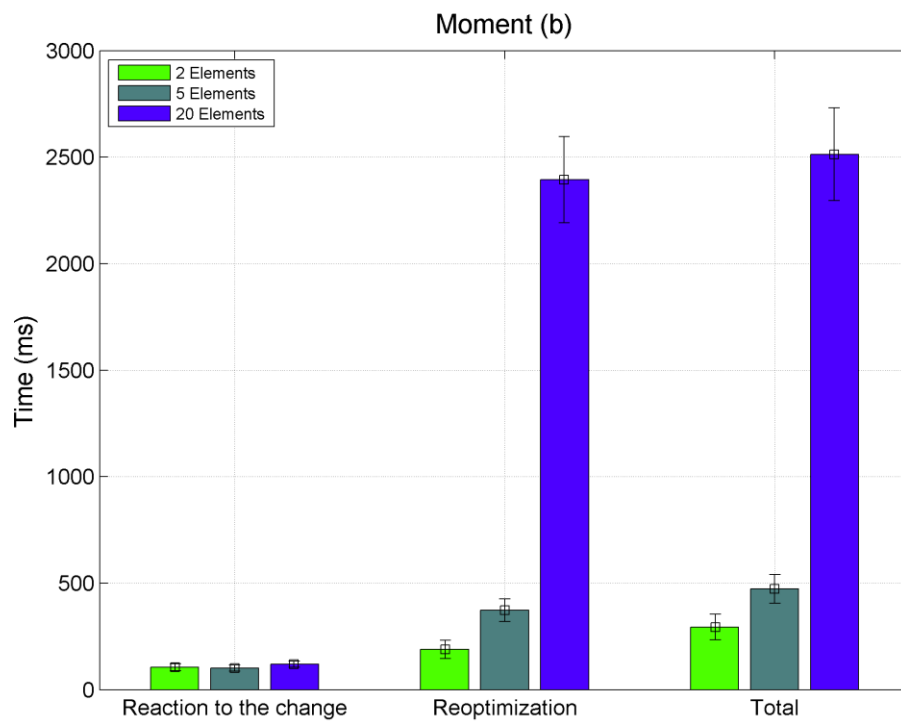


Figure 5.10 – Time to react to a link removal (substrate network)

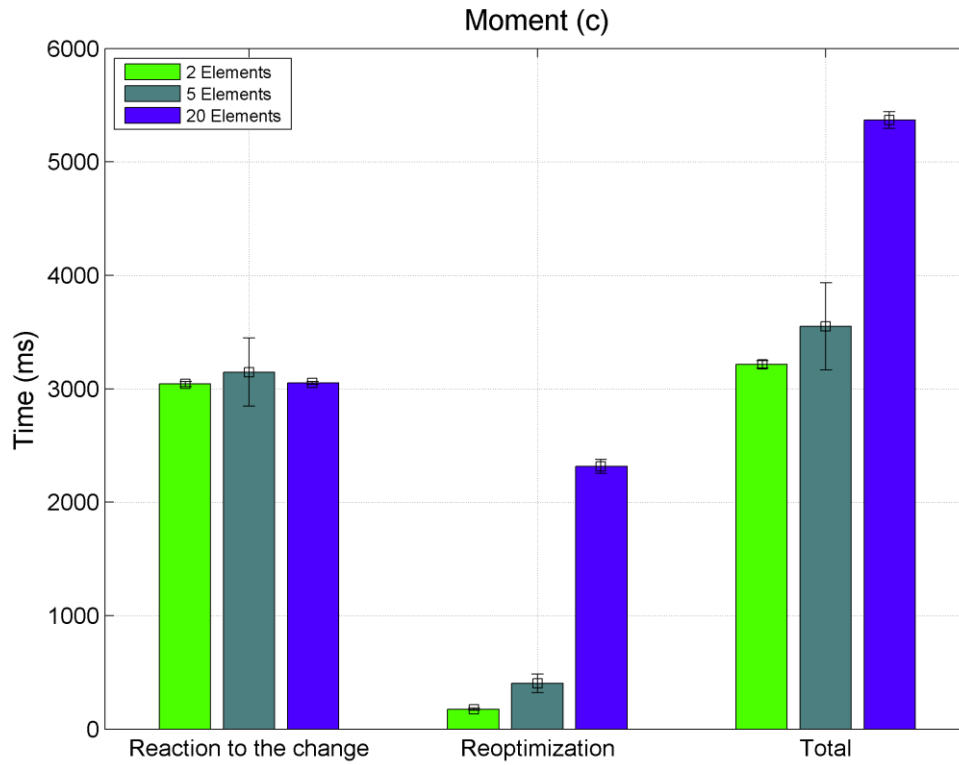


Figure 5.11 – Time to react to a link addition (substrate network)

Related with the operation modes, we conclude that the full mesh form consumes lower time, but is restricted for the cases that we want to have connections in the both ways between the elements.

In summary, for the multicast form, a connectivity service that is translated in one service takes approximately 0.460 seconds to be received, treated and activated. In the case that the service results in four flows, it takes approximately 1.260 seconds. For the last service, which is translated into 19 flows, it takes approximately 5.8 seconds to be activated. These times will be affected if the size of the substrate network increases. This is related with the policy used for finding the path for the flows (*Dijkstra* that gives the shortest path). When there is a loss of a link in the substrate network, our system only modifies the flows that are affected. However, when there is an addition of a link in the substrate network, our system tries to optimize all the flows according to the path finding policy, which is the shortest path. In the cases that a shorter path is found, that flow will be modified immediately. The difference of performance for both cases is in the identification that the substrate network has changed, that is quicker for the loss of a link than the addition of a link. However, when the number of flows activated in the substrate network is larger, it is expected that the reoptimization process due to the addition of elements will have longer time consumption than the cases when there is a loss of elements.

5.5. Conclusion

This chapter gives a perspective of the performance of our SDN framework for the activation of connectivity services. More than that, it also shows the response times needed to react to changes in the substrate network, in order to let those current services active, or to improve the path selection of their flows according to the internal policies (shortest path).

This module has three different features: receives connectivity services and performs their translation into flows; finds a path solution and activates those flows; and reacts in an intelligent way to problems noticed in the substrate network. This module does more than the ordinary routing with the common routing protocols, because it tries to optimize the current flows and also reacts to problems that regular switches protocols cannot answer.

6. Conclusions

6.1.1. Final Conclusions

The work presented in this Dissertation had two main objectives: to develop a mapping algorithm able to map in an integrated manner cloud and network resources with the goal of maximizing the profits of the operator and study the impact of the developed link reoptimization algorithm; and the creation of an intelligent component that receives, activates and manages services on an Openflow network according to the results of the first part.

The starting point to the implementation of the mapping algorithm and the link reoptimization was the mapping algorithms presented in the state of art, by Monteiro [62] and Soares [63].

In chapter 3, *Cloud Networking Mapping Algorithm*, it was proposed several modifications in order to maximize the embedding of VIs. Moreover, we redirect the focus of the base mapping algorithm from the load balance policy to take in consideration also the link usage of the physical infrastructure. With a stronger list of candidates and some modifications to the parameters that calculate their potential, we improved the embedding of the VI at the cost of lower occupation resources and power consumption of the physical infrastructure. Also, we introduced assurance mechanisms to respect the conditions of work. The load balance policy was used to differentiate candidates that would be at the same group of the number of links consumption. In those cases, the lower stressed candidates would be chosen. Here, it was performed an individual study for every enhancement introduced for the online version of the mapping problem.

The chapter 4, *Link Reoptimization algorithm*, studied the impact of the reoptimization of virtual links, while virtual nodes are kept untouched. It was possible to improve the acceptance of VIs, which turns out in more revenue to the physical infrastructure provider, and to improve the physical infrastructure utilization (link capacity, node usage and link usage). In this part different approaches were compared. With the division in bandwidth groups, the aspect that was largely optimized was the occupation levels of the physical infrastructure, because it was possible to better map the window of links that are formed by a set of old and new VIs (using lower number of physical links in a global point of view). Naturally, with a better use of the physical infrastructure resources, we would obtain better results in terms of acceptance, which will result in a larger revenue for the operator, but it is noticeable a difference of performance of those aspects (the embedding of VIs has increased less than the expected). Our considerations in this

last point go to the path finding algorithm used in the reoptimization, which probably did not potentiate the method introduced.

Finally, an SDN compliant module was developed and presented in chapter 5. This module, entitled *Service Manager*, is responsible for receiving connectivity service requests and managing them over an OpenFlow network. This is an initial approach, which works with small number of substrate nodes, and for that the *Dijkstra* algorithm is appropriate for the path finding of the flows (shortest path), as it avoids major problems. Furthermore, it allows two ways of work: full mesh that creates a bidirectional connection between all the elements two in two; and the multicast that creates an unidirectional connection between the source element and a set of target elements. Finally, the *Service Manager* reacts to changes in the substrate network with the objective to keep the connectivity services active and optimized for the path of the flows it is concerned.

6.1.2. Future Work

Starting with the mapping algorithm, still some work can be done in order to improve its performance, such as the creation of a better metric to calculate the potential of the physical nodes in the selection process. A suggestion is to consider only the paths and stress of the physical links. The bandwidth of the physical substrate is the main constraint, and probably should be better exploited. In future studies we also propose to normalize the elements that compose the potential. The decision of candidates would then work always in the same range of values. A point that was not completely explored in this work was the interdependence of the candidates to a higher level than the immediate neighbors. Other aspect that could be interesting to see implemented in this mapping algorithm is a green mapping objective, giving priority to the physical nodes and links already in use.

Related with the link reoptimization, we think the path finding algorithm is not well adjusted to reach all the potential of our link reoptimization structure. More than that, it could be interesting to study the conditions of the substrate, stress levels according to the substrate characteristics, in which it should be profitable to perform the reoptimization in spite of stipulating a fixed period (X in X iteration). More than that, with that study of the substrate condition, it could be better to mix different strategies into the reoptimization algorithm instead of just fixing one. This dynamism could tackle problems such as main links overhead.

The *Service Manager* module is in an early stage and can be further improved. At the moment, it is restricted to connectivity services with well defined elements, and it does not allow the connectivity services to specify the type of protocols to implement. Besides that, the module could take more advantage of the alarms information and allow the setting of priority levels to the flows (i.e. in the cases that some specific link is overwhelmed, the *Service Manager* could decide to move the flows with lower priority until the problem is fixed). Another solution could be to locate the flow that is creating that problem and deal with it directly, instead of changing flows that are working normally. This last solution requires a more complex level of intelligence for the module. Moreover, the internal performance of the module, specially the interactions with the

database, can be better exploited. Instead of making all the functions sequential, this module can perform all the updates of the database tables at once. This functionality is supported by the MySQL and can easily improve the performance of the module.

Bibliography

- [1] P.Mell and T.Grance, "Recommendations of the National Institute of Standards and Technology", September, 2011. <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- [2] STAMFORD, Conn. Gartner "Highlights Five Attributes of Cloud Computing", June, 2009. <http://www.gartner.com/newsroom/id/1035013>
- [3] J.Gabrielsson, O.Hubertsson, I.Más and R. Skog, "Cloud computing in telecommunication", January, 2010. http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2010/cloudcomputing.pdf
- [4] Amazon Elastic Computing Cloud, <http://www.aws.amazon.com/ec2>. Retrieved 1 June 2013.
- [5] Cloud Hosting, Cloud Computing and Hybrid Infrastructure from GoGrid, <http://www.gogrid.com>. Retrieved 1 June 2013.
- [6] Google App Engine, <http://code.google.com/appengine>. Retrieved 1 June 2013.
- [7] Windows Azure, <http://www.microsoft.com/azure>. Retrieved 1 June 2013.
- [8] Salesforce, <http://www.salesforce.com/platform>. Retrieved 1 June 2013.
- [9] Dedicated Server, Managed Hosting, Web Hosting by Rackspace Hosting, <http://www.rackspace.com>. Retrieved 1 June 2013.
- [10] Pertino, <http://pertino.com/>. Retrieved 1 June 2013.
- [11] Amazon Web Services, <http://www.aws.amazon.com>. Retrieved 1 June 2013.
- [12] OpenStack: The Open Source Cloud Operating System, <http://www.openstack.org>. Retrieved 1 June 2013.
- [13] OpenStack: "OpenStack User Stories", <http://www.openstack.org/user-stories/>. Retrieved 1 June 2013.

- [14] OpenStack: “Companies Supporting The OpenStack Foundation” <http://www.openstack.org/foundation/companies/>. Retrieved 1 June 2013.
- [15] OpenStack: “OpenStack End Users”, <http://docs.openstack.org/folsom/openstack-object-storage/admin/content/figures/openstack-logical-arch-folsom.jpg>. Retrieved 1 June 2013
- [16] CloudStack: Open Source Cloud Computing, <http://www.cloudstack.apache.org/>. Retrieved 1 June 2013.
- [17] VMware. <http://www.vmware.com/>. Retrieved 1 June 2013.
- [18] Kernel Based Virtual Machine. http://www.linux-kvm.org/page/Main_Page. Retrieved 1 June 2013.
- [19] XenServer, citrix. <http://www.citrix.com/products/xenserver/overview.html>. Retrieved 1 June 2013.
- [20] Citrix: “Leading Telecommunications Companies Choose Apache CloudStack to Power their Cloud Services”, October, 2012. <http://www.citrix.com/news/announcements/oct-2012/leading-telecommunications-companies-choose-apache-cloudstack-to.html>.
- [21] Apache CloudStack 4.0.0-incubating: “CloudStack Instalation Guide”, http://cloudstack.apache.org/docs/en-US/Apache_CloudStack/4.0.0-incubating/html-single/Installation_Guide/. Retrieved 1 June 2013.
- [22] OpenNebula: Open Source Data Center Virtualization, <http://www.opennebula.org/>. Retrieved 1 June 2013.
- [23] OpenNebula: Enterprise Private Clouds and Datacenter Virtualization, <http://www.opennebula.org/users:users>. Retrieved 1 June 2013.
- [24] Z.Kerravala: “Why Cloud Computing Needs a Cloud-Intelligent Network”, April, 2012. http://www.cisco.com/en/US/prod/collateral/routers/ps10537/zk_white_paper_cloudverse.pdf
- [25] Rosen, E.: “Multi-protocol label switching (mpls) architecture”; rfc 3031, 2001. <http://tools.ietf.org/html/rfc3031>
- [26] K. Kompella and Y. Rekhter, “Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling”, *IETF RFC 4761*, January, 2007.
- [27] M. Lasserre and V. Kompella, “Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling”, *IETF RFC 4762*, January, 2007.
- [28] J. Soares, R. Monteiro, M. Melo, S. Sargento and J. Carapinha, “The Cloud inside the Network: a virtualization approach to resource allocation”, in: H. Mouftah and B. Kantarci (Eds.), *Communication Infrastructures for Cloud Computing: Design and Applications*, IGI Global, 2013.

- [29] IBM, "Networking for cloud computing", white paper, April, 2013.
<http://public.dhe.ibm.com/common/ssi/ecm/en/icw03005usen/ICW03005USEN.PDF>.
- [30] Cisco, "Network Virtualization—Path Isolation Design Guide", February, 2009.
http://www.cisco.com/en/US/docs/solutions/Enterprise/Network_Virtualization/PathIsol.pdf
- [31] N. Chowdhury and R. Boutaba, "Network Virtualization: State of the Art and Research Challenges", *Communications Magazine, IEEE*, vol.47, no.7, pages 20–26, July, 2009.
- [32] GENI: Global Environment for Network Innovations. <http://www.geni.net>. Retrieved 1 June 2013.
- [33] N. Feamster, L. Gao, and J. Rexford, "How to Lease the Internet in your Spare Time", *SIGCOMM Computer Communication Review*, vol. 37, no. 1, 2007, pages 61–64.
- [34] 4ward, The 4WARD project, <http://www.4ward-project.eu/>. Retrieved 1 June 2013.
- [35] NICIRA: Network Virtualization Platform. <http://www.nicira.com/en/network-virtualization-platform>. Retrieved 1 June 2013.
- [36] HP: "OpenFlow: Enabling technology for software-defined networking".
<http://h17007.www1.hp.com/br/pt/solutions/technology/openflow/index.aspx>. Retrieved 1 June 2013.
- [37] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks". *ACM SIGCOMM Computer Communication Review*, vol.38, no.3, pages 105–110, 2008.
- [38] Pox. <http://www.noxrepo.org/pox/about-pox/>. Retrieved 1 June 2013.
- [39] Jaxon: java-based openflow controller. <http://jaxon.onuos.org/>. Retrieved 1 June 2013.
- [40] Simple Network Access Control (SNAC). <http://www.openflow.org/wp/snac/>. Retrieved 1 June 2013.
- [41] Floodlight, an open sdn controller. <http://oodlight.openflowhub.org/>. Retrieved 1 June 2013.
- [42] Ryu. <http://osrg.github.com/ryu/>. Retrieved 1 June 2013.
- [43] M. Mendonca, B. Nunes, X. Nguyen, K. Obraczka and T. Turlitti, "A Survey of Software-Defined Networking: Past, Present and Future of Programmable Networks", May, 2013
http://hal.inria.fr/docs/00/83/50/14/PDF/bare_jrnl.pdf

- [44] K. Bouyoucef, I. Limam-Bedhiaf, and O. Cherkaoui: "Optimal allocation approach of virtual servers in cloud computing". In *Next Generation Internet (NGI), 2010 6th EURO-NF Conference on*, pages 1–6, June.
- [45] M.J. Csorba, H. Meling, and P.E. Heegaard: "Ant system for service deployment in private and public clouds". In *Proceeding of the 2nd workshop on Bio-inspired algorithms for distributed systems, ACM*, pages 19–28, 2010.
- [46] F. Ma, F. Liu and Z. Liu: "Distributed load balancing allocation of virtual machine in cloud data center". In *Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on*, pages 20–23, June, 2012.
- [47] Y. Zhu and M. Ammar: "Algorithms for assigning substrate network resources to virtual network components". In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*. Proceedings, pages 1–12, 2006.
- [48] J. Nogueira, M. Melo, J. Carapinha, and S. Sargento: "Virtual network mapping into heterogeneous substrate networks". In *IEEE Symposium on Computers and Communications (ISCC)*, pages 438–444, June 2011.
- [49] N. Chowdhury, M. Rahman, and R. Boutaba: "Virtual network embedding with coordinated node and link mapping". In *INFOCOM 2009, IEEE*, pages 783–791, 19-25 April 2009.
- [50] J. Lischka and H. Karl: "A virtual network mapping algorithm based on subgraph isomorphism detection". In *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 81–88, New York, NY, USA, 2009.
- [51] M. Melo, J. Carapinha, S. Sargento, L. Torres, T. Nga, U. Killat, and A. Timm-Giel: "Virtual network mapping - an optimization problem". In *MONAMI - 3rd International ICST Conference on Mobile Networks & Management*, 2011.
- [52] M. Yu, Y. Yi, J. Rexford and M. Chiang: "Rethinking virtual network embedding: substrate support for path splitting and migration". In *SIGCOMM Computer Communication Review*, Volume 38, Issue 2, pages 17–29, April, 2008.
- [53] I. Houdi, W. Louati, W. Ameer and D. Zeghlache, "Virtual network provisioning across multiple substrate networks". In *Computer Networks*, Volume 55, Issue 4, pages 1011–1023, 10 March 2011.
- [54] W. Jiang, R. Zhang-Shen, J. Rexford and M. Chiang: "Cooperative content distribution and traffic engineering in an ISP network". In *SIGMETRICS Performance Evaluation Review*, Volume 37, Issue 1, pages 239–250, June, 2009.
- [55] N. Roy, J.S. Kinnebrew, N. Shankaran, G. Biswas and D.C. Schmidt: "Toward effective multi-capacity resource allocation in distributed real-time and embedded systems". In

Object Oriented Real-Time Distributed Computing (ISORC), 11th IEEE International Symposium on, pages 124–128, May, 2008.

- [56] L. Li and M. Tang: “Novel spectral method for server placement in cdns”. In *Advanced Computer Theory and Engineering (ICACTE), 3rd International Conference on*, Volume 6, pages 197–199, August, 2010.
- [57] T. Enokido, A. Aikebaier, M. Takizawa and S. Deen: “Energy-Efficient Server Selection Algorithms for Network Applications”. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), International Conference on*, pages 159–166, 2010.
- [58] T. Enokido, A. Aikebaier, M. Takizawa and S. Deen: “Power Consumption-Based Server Selection Algorithms for Communication-Based Systems”. In *Network-Based Information Systems (NBIS), 13th International Conference on*, pages 201–208, 2010.
- [59] B. Kantarci and H. Mouftah: “Minimizing the provisioning delay in the cloud network: Benefits, overheads and challenges”. In *Computers and Communications (ISCC), IEEE Symposium on*, pages 806–811, 1-4 July, 2012.
- [60] B. Kantarci and H. Mouftah: “Optimal reconfiguration of the cloud network for maximum energy savings”. In *Cluster, Cloud and Grid Computing (CCGrid), 12th IEEE/ACM International Symposium on*, pages 835–840, 13-16 May, 2012.
- [61] B. Kantarci and H. Mouftah: “Overcoming the energy versus delay trade-off in cloud network reconfiguration”. In *Computers and Communications (ISCC), IEEE Symposium on*, pages 53–58, 1-4 July, 2012.
- [62] R. Monteiro, “Creation and reconfiguration of virtual networks in the operator perspective”, Master Thesis, *Department Electronic Telecommunication and Informatics, University of Aveiro*, Aveiro, Portugal, 2011.
- [63] J. Soares, J. Carapinha, M. Melo, R. Monteiro and S. Sargento, "Resource allocation in the network operator's cloud: A virtualization approach," *2012 IEEE Symposium on Computers and Communications (ISCC), 2012 IEEE Symposium on Computers and Communications (ISCC)*, 2012, pages 800–805.